Enterprise Scrum Definition: Agile Management for the 21st Century

Authored, Developed and Sustained

by

Mike Beedle

Enterprise Scrum Inc.

August 4, 2014

Release 1.02

Acknowledgements

First and foremost, I would like to thank Jeff Sutherland for inventing what we now know as modern Scrum in the fall of 1993. Without his work and inspiration Enterprise Scrum will not be possible.

I would also like to thank Ken Schwaber for all his early contributions in defining, documenting, explaining and popularizing Scrum from the very early days dating back to 1994. His determination, understanding and vision of Scrum have been instrumental on making Scrum successful worldwide.

I would also like to thank Jim Coplien for the early introduction of organizational patterns, which are one of the foundations of modern Scrum. Without organizational patterns there would be no modern Scrum, and little hope to expand the initial knowledge base into Enterprise Scrum.

Lastly, I want to thank all of those of you – customers, Enterprise Scrum students, conference attendees and fellow Scrum trainers and coaches:

1) For all the interesting conversations and shared insights
2) For all the feedback on presentations and class materials
3) For all their questions about problems on the field over the years, and
4) For the opportunity to help you in a myriad ways to implement Enterprise Scrum in the trenches

To all the countries, skies and cities that hosted my writings, in their restaurants, airports, cafes and bars – THANK YOU:

Lake Forest, Lviv, Munich, Frankfurt, Warsaw, Krakow, Barcelona, Brussels, Copenhagen, Elmhurst, Oslo, Chicago, Tisvildeleje, Lansing, Buenos Aires, Santiago, Atlanta, Cochabamba, Boston, Lima, Philadelphia, Prague, Washington DC, Montevideo, Dallas, Fort Lauderdale, Miami, London, Toronto, New York, San Francisco, Phoenix, Shanghai, Seattle, Key West, Nassau, Mallorca, San Diego, Kansas City, Guadalajara, Paris, Tokyo, Nowa Jablona, Glogow, Monterrey, Berlin, Mexico City and Madrid – where this document was finished to its first version.

I hope this is useful to all present and future users of Scrum so that they can use Scrum for:

1) more Business-like purposes, 2) in a Generic way, and 3) scale it; if necessary.

Change Log

1.02 Added more business references and linked Enterprise Scrum to them, such as Running Lean, Business Model Generation, Beyond Budgeting, Smart Tribes, Tribal Leadership, etc. Added Architecture-related parameters. Abstract 4 high-level Enterprise Scrum patterns: Scrum Team, True Business Value, Plan by Measurement, Adapt through Improvement Cycles.

1.01 Added more scaling references. Cross Functional Skill Matrix.

1.0 Base Definition. Enterprise Scrum naming conventions.

Business-Orientation: Multiple nested Improvement Cycles, insertion of techniques, calculations: budgets, schedules, fixed-date, risk-management, other cumulative metrics.

Genericity: business value, DOR, DOD, VLI types, Metrics and charts, generalized velocities, etc.

Scaling: meeting options, rules and participants, parent, contributors, dependOn, dependsOnUs, value list parent, global and local velocity, cumulative metrics, etc.

Contents

1. What is Enterprise Scrum?

Faster Change

The need for faster more agile management is higher every year. John Kotter, which wrote the book *Leading Change* [Kotter1], has concluded that the rate of change in business is growing exponentially [KotterForbes]. Evidence for this argument, he says, is shorter product and service life cycles and therefore the need for faster and better-targeted innovation.

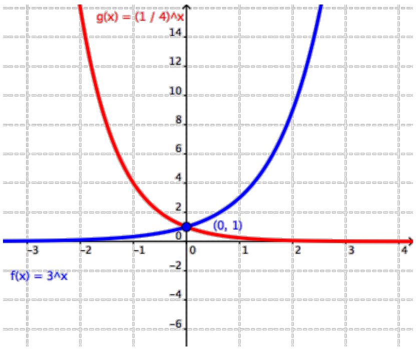


Figure 1 Exponential Change means Exponential smaller reaction time

This is also supported by the survey in *Best Practices in Product Innovation: What Distinguishes Top Performers* in 2011 by Cooper, Edgett and Kleinschmidt [CEK], that points to an average of 45% of profit and revenue come from products and services invented in the last 5 years, and 70% of profit and revenue come from products and services invented in the last 5 years for the top 20% performers. When Nonaka and Takeuchi wrote their paper first describing Scrum, *The NEW new Product Development Game* [NonakaTakeuchi], the percentage of profit and revenue for products invented in the last 5 years was measured in the 1970s to be 20% on the average, with a prediction of 33% in the 1980s. There is also an interesting dynamic that applies when companies achieve market share with a winning product: they get themselves into the Innovator's Dilemma. Is it better to continue to compete and dominate in a validated market through "sustaining innovation" – bettering the existing products, or to pursue more "breakthrough innovation"? [Christenssen], [Kanter], [Kaplan]. The answer is they need to do both, of course: protect revenue and lead through innovation.

> **If your company is not innovating today, five years from now is likely that it would lose 45% of its revenue and profits on the average.**

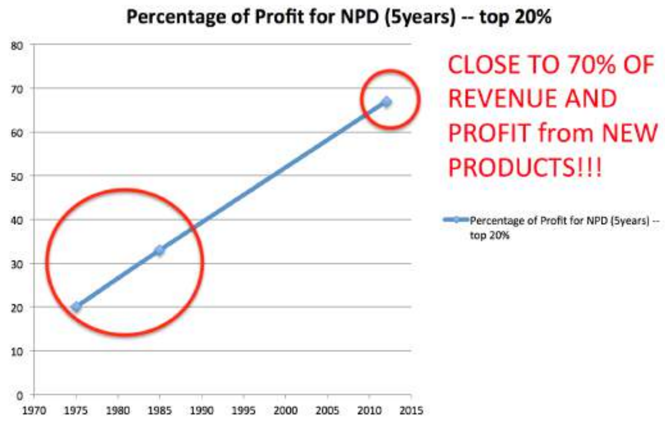


Figure 2. Revenue and Profit for top 20% competitors from NEW products in 5 years

The consequences of the increasing exponential business rate of change are very many:

- If business change in general is growing exponentially, this means that as company managers we have an **exponentially shorter reaction time to adapt, and an exponentially shorter predictable time horizons**.
- As company managers, we have much more market information to process in a shorter time: **we need to process ever-higher amounts of information of ALL kinds with shorter reaction times** (competitors, suppliers, technology, etc.)
- This leads to stronger sharper competition – **products and services either make it or break it faster with ever decreasing product and services lifecycles**
- **To compete we need to innovate and delight faster** – we need to have good coherent visions of what the customer wants or needs and **bring NEW products and services that can delight them faster to market**
- This implies that **we must process Customer Feedback, Market Feedback and Technology changes faster**
- We need to have more certainty in development and **make predictable NEW product announcements,** so that we can pre-sell our products and services
- As competition stiffens we need **better managed Products/Services portfolios** that cut down waste of non-profitable, obsolete or non-competitive products or services
- One of the consequences of the above, is that **our products and services may need to work or fail as fast as we can**, so that we can identify which products or services we will keep in our portfolio

New Product Efficiency

As if this wasn't enough pressure with the sheer "accelerated rate of business change", there is a wide gap between among players within an industry.

Arthur D. Little which defines "New product efficiency" as New Products Sales divided by R&D investment, in their *Innovation Excellence Study* [ArthurDLittle], reveals that the best innovators are 12X more productive. **This means "efficient innovators" get twelve times more sales for the same R&D investment.**

But because a large percentage of both revenue and profit comes from NEW products and services, as we saw above, **innovation efficiency is critical to the company success**, because the world has turned into an **innovation competition** within every industry and market segment.

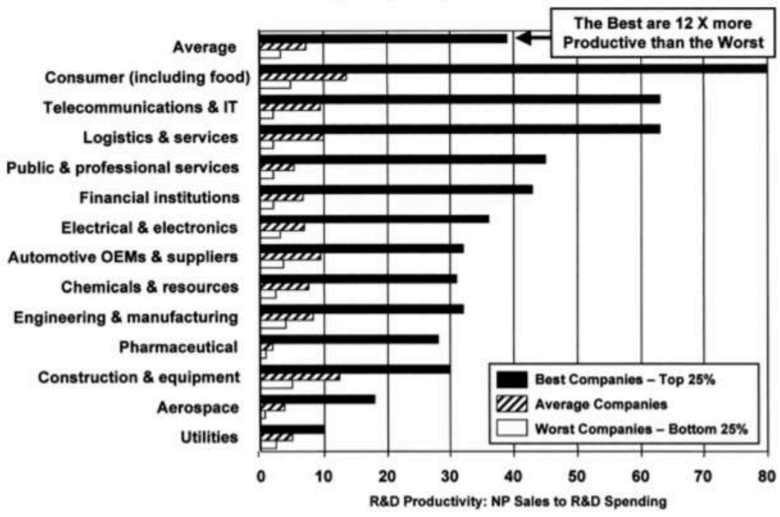


Figure 1 .— NPD productivity varies greatly among companies, with huge differences between the best and worst Companies in each industry. Source: A. D. Little Innovation Excellence Study (5). Source: A.D. Little Innovation Excellence Study (5).

Figure 3. New Product Efficiency by Industry

What are the consequences of this gradient in New Product Efficiency?

- **Winners innovate faster with less percentage investment**
- Winners optimize ROI on new product and service development, **investing the lowest possible to get the maximum sales, profits and/or market share**.
- **Innovation takes place anywhere in the process**: strategy, marketing, technology, customer feedback.
- **Winners often get larger market share faster** (market share is not necessarily a measure of profitability but is still a measure of competitive advantage)
- **Winners have a higher revenue and profit percentage from NEW products and services in shorter times**
- **New Product Development efficiency differences can be very large 1200%** among slow and fast innovators (new product sales vs. R&D investment)
- **Product Development Efficiency is good measure of business SUCCESS, specially in today's environments** – we want to generate the highest profits with the lowest re-investment in the shortest possible time

A New Kind of Management

As we reached our conclusions from both the exponential rate of business change and the New Product Efficiency sections, one clear picture emerges:

> **To manage and lead our companies to be successful in the 21st century we need faster, more efficient and customer-driven innovation to obtain higher profits and revenue** – just like Nonaka and Takeuchi concluded in their HBR paper back in 1986

Except, in today's environment of more complex business models, ***we need to reinvent our entire companies not only one product at a time:***

> ***the product is the company itself.***

We can use our same trusted technique for rapid innovation – Scrum, just as Nonaka and Takeuchi concluded, but now for the entire enterprise. Or to give it a name, we need **Enterprise Scrum**.

Some people have already started to practice and write about this new style of management. For example, Steve Deming – a Scrum Alliance board member, wrote the book Radical Management [RadicalManagement].

Empirical Process Management

Where is most change and uncertainty come from in an enterprise? Change is everywhere, and it can come from very many different directions, but where does the most critical change come from? The process answer to this question is: from business processes that are open information systems.

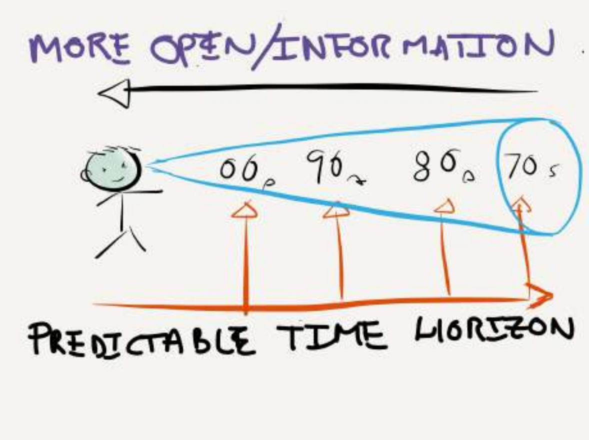


Figure 4. Development-Like processes are Open Information Systems

The processes of an enterprise can be broadly divided into production (or defined) processes and development (or empirical) processes. Defined processes are processes that can be changed at a slower pace while development-like processes need faster, higher-frequency, often larger and more pointed changes in time. Defined processes execute with all the steps known in advance, while development-like or empirical processes execute while still admitting new information into the process [ProcessControlTheory].

Defined processes are processes like production, manufacturing, accounting, payroll, or customer service. A manufacturing process can be improved but the steps involved as it executes in that version or instance of the process are known.

Production-like processes should be optimized through **Lean Production** techniques – Lean Principles and Lean Techniques applied to production-like processes [Ohno], [Wommanck1], [Wommanck2], [Wommack3]. Most Global 5000 Companies have been optimizing their production-like processes using these techniques for nearly 30 years, and longer in Japan.

Development processes on the other hand, are processes like strategy, marketing, product/service development or R&D; that can admit new information leading to new steps as the process executes. In these processes major changes can occur even within a few days with a new partnership, a new product offering by a competitor, or a new discovery in R&D. In parallel to Production-like processes, Development-like processes should be optimized with **Lean Development** techniques.

> ***But the fact is, development-like processes, which will give in fact most of the top-line and competitive advantage to firms in the 21st century, have not yet been optimized hardly at all.***

Why? Because in the 20th century, most Global companies spent most of their efforts optimizing their production-like processes, and because it was not very clear how these development-style processes were different from defined processes, or how to optimize the development-like processes. Knowing and understanding this difference is understanding the power of Scrum:

> ***Scrum is a compact, easy-to-understand, and well-proven Lean Development technique that can provide empirical process management for any business process.***

Any process instance within the enterprise that has a development-style flavor can be better served by using Scrum than by a defined style process. A development style flavor process instance is a process instance where there is high uncertainty because:

1) NEW information is still flowing into the system.
2) changes happen frequently (business, market or technology),
3) understanding existing or new information is hard and misunderstandings are easy,
4) people and teams change over time,

Empirical process management assumes an **open information system**, where we must adapt as new information comes in, or as our understanding of the up-front conditions becomes better. Empirical process management is based on transparency, inspection and adaptation:

Transparency – so that everything is visible and understood through a common language including the domain and basic Scrum definitions like DOR or DOD
Inspection – so that we can determine through agreed measurements the current state in terms of a common understanding of the goals and the standards
Adaptation – so that we can improve

For example, the true state of a company must be **transparent**, so that we can **inspect** its current market position, and then **adapt** to improve its strategic position.

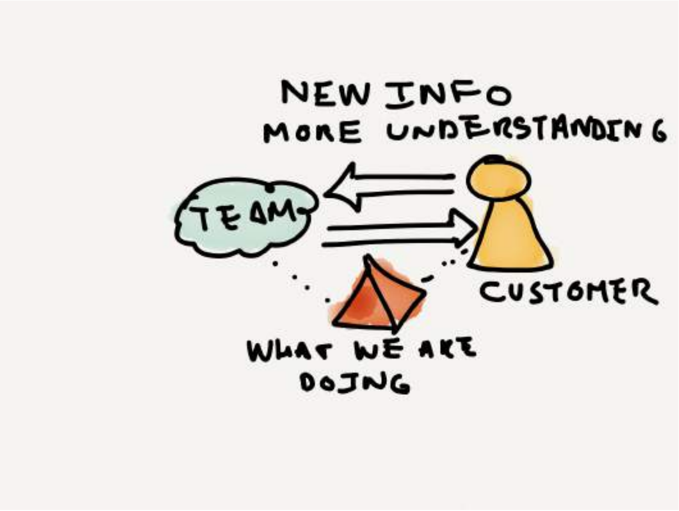


Figure 5. New Information and more understanding make Development-like processes uncertain

As such, processes like Strategy, Marketing, Product Development, Software Development, Basic Research, Business Process Re-Design are better managed with Scrum (empirical process management) rather than by traditional defined-process style project or process management. These are in fact the top-line processes that will give Agile companies competitive advantage in the 21st century.

As we will see, collections of defined process instances also have an empirical flavor because of the statistics of the defined processes, so Scrum is also a very useful framework for managing collections of defined processes.. Moreover, because process redesign is a creative process, Scrum can also be used effectively for this purpose.

These are not just “good ideas”, there are many companies that are already taking advantage of this style of management to adapt faster and out-innovate its competitors.

The Scrum Framework

Scrum is a process framework for product development that delivers the most business value in the shortest possible amount of time. Because Scrum is a framework, we can fit many techniques for *product development.* We will not spend much time describing basic Scrum [ScrumGuide]. Instead, we assume that the reader already knows basic Scrum and that he or she wants to use Scrum for more business purposes, in a generic way, or that you are interested in scaling its application.

> **The purpose of Enterprise Scrum is not to redefine any aspects of Scrum. Rather, the purpose of Enterprise Scrum is to provide an expanded and more detailed framework foundation of Scrum so that we can use Scrum for business, generic or scalable purposes.**

Enterprise Scrum is not a different version of Scrum, or a different flavor of Scrum; Enterprise Scrum is simply a powerful detailed abstraction of Scrum. We strongly encourage the reader to read the Scrum Guide as the only place where Scrum is defined [ScrumGuide]. Scrum is also described in detail at the Agile Atlas [AgileAtlas].

Brief history of Agile and Scrum

The origin of Agile ideas in business started with the creation of the Agile Consortium and the publication of *The 21 century manufacturing enterprise strateg*y in 1991 [Nagel1] and described in detail in the book *Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer* [AgileCompetitors]. In software these ideas are continued and unified by the creation of the *Agile Manifesto* [AgileManifesto]. There were many early ideas that pointed that this was a good direction from the very early ways. See for example, *Mythical Man Month* [MMM], *Peopleware* [PeopleWare], and *Wicked Problems, Righteous Solutions: A Catalog of Modern Engineering* by Peter DeGrace, Leslie Hulet Stahl.

The origin of the Scrum ideas comes from product development in Toyota [Ohno], [Liker1], [Wommack1] i.e. Toyota Product Development System (TPDS) [Liker2]. Lean and Agile product development eventually made it into other Japanese companies, such as Honda, Fujitsu, Cannon, etc. See Nonaka and Takeuchi's *The NEW new product development game* [NonakaTakeuchi].

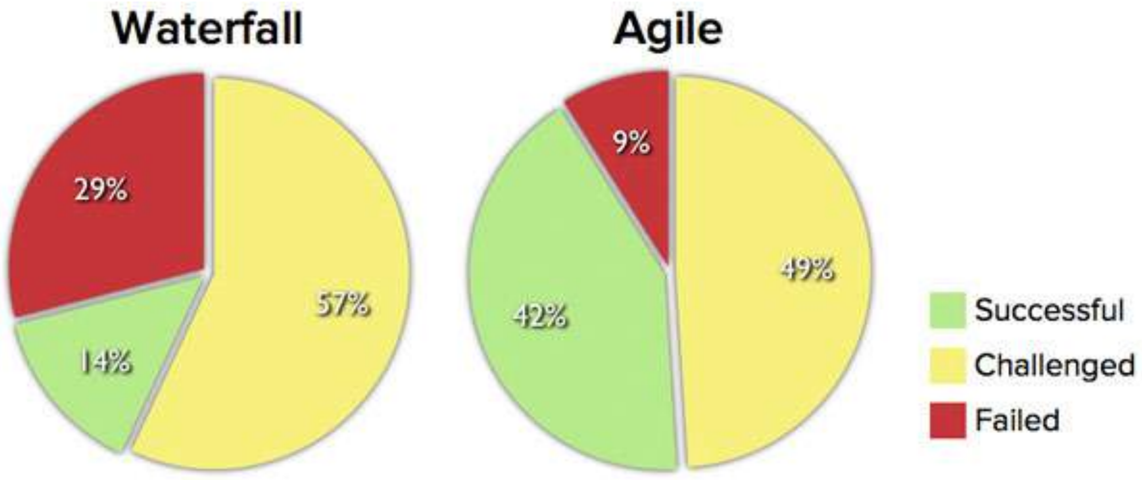
TPDS [Liker2], and other similar development styles emerged after using for the purposes of development some of the principles and techniques of the Toyota Production System (TPS) [Ohno], which itself emerged from the TWI (Training Within Industry program) [TWI].

However, Jeff Sutherland invented "modern Scrum", the basis for Enterprise Scrum, in 1993 by putting together three concepts together:

1) the ideas from Nonaka and Takeuchi explained in *The NEW new product development game* [NonakaTakeuchi].
2) Organizational Patterns, recurring best practices in successful teams first documented by Jim Coplien after analyzing what hyper-productive teams were doing [Coplien] [OrgPatterns], and the
3) Subsumption Architecture, the concepts of artificial intelligence discovered by Rod Brooks [Brooks1], [Brooks2].

The first book on "modern Scrum" (and Agile) was *Agile Software Development with Scrum* [AgileSoftwareDevelopment], but now there are many other good books on the subject.

As we will see, the understanding of this history will be very important to understand Enterprise Scrum, as Enterprise Scrum is a continuation of these original ideas, but now extended to the enterprise as a whole. There have been some early attempts to create frameworks for agile companies, see for example, *cOOherentBPR: A pattern language to build Agile organizations* [Beedle-cOOherentBPR-1997], or *Enterprise Architecture Patterns: Building Blocks of the Agile Company* [Beedle-EnterpriseArchitecturePatterns-1998].



Source: The CHAOS Manifesto, The Standish Group, 2012.

Figure 6. Standish Group, CHAOS Manifesto results 2012

Today we know Scrum is better than other styles of management in software development – see for example the Standish Group's CHAOS report [StandishGroup]; but we are starting to get empirical evidence, that Scrum is just better management for any business process.

Enterprise Scrum High-level definition

So, if we need to agilize everything – empirically manage everything, and allow innovation anywhere within the enterprise, how can we use Scrum for this purpose?

Enterprise Scrum is about **extending, generalizing, and parameterizing regular Scrum** so that Scrum can work:

1) **more Business-like** (business cycles, releases, business metrics, adding business techniques).
2) **In a Generic way, *at different levels or the organization*** (executive, middle management, program/project level, etc.), and ***for different business process*** (company management, marketing, sales, product development, software development, research, audit, etc.),
3) in a **Scalable** way, by having multiple Scrum teams working together a) independently, b) in cooperation for the same purpose – we defined this as scaling mode, or c) with different purposes but in cooperation – we defined this as networking mode

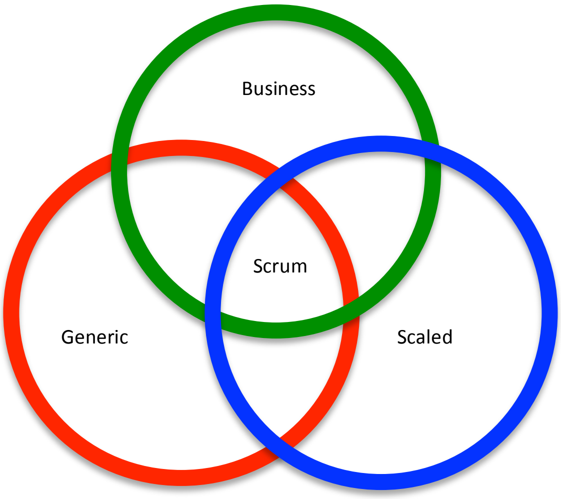


Figure 7. Enterprise Scrum: business-like, generic, scaled Scrum

Therefore, we formally define Enterprise Scrum as:

> ***Enterprise Scrum is the extended, generalized and parametrized application of Scrum for 1) more business purposes, 2) in a generic way -- at different levels in the organization and for different business processes, and 3) in scaled mode (distributed and/or cooperative).***

In Chapter 2, *Enterprise Scrum Definition*, we show the exact definition of Enterprise Scrum, describing how through parameters we can customize Scrum to be business-like, generic and scaled. The Enterprise Scrum Definition also includes some guidance and examples as to how to implement Enterprise Scrum. In chapter 3, we provide a full Enterprise Scrum example.

Patterns

These extensions and parameters come as conclusions from observing very many instances of Scrum in different domains and industries since 1995. Therefore:

> ***Enterprise Scrum emerged from the observation of thousands of Scrum projects and processes in different domains and different industries where people in the trenches had already informally customized Scrum for the purpose of doing Agile Management.***

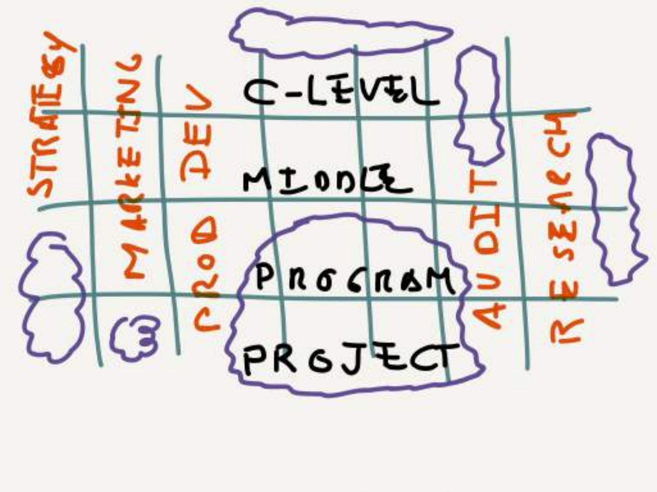


Figure 8. Scrum is being applied EVERYWHERE.

There are thousands of examples, ranging from startup management, through mid-sized companies, and Global 5000 companies. The definition of Enterprise Scrum emerged from the trenches -- from all the instances of Scrum for different purposes that I have either implemented, observed, heard about, discussed formally or informally, consulted on, taught about, or mentored on.

So the process by which these patterns have been found and validated is by observing the world and mining patterns, and then applying the patterns and verifying that the patterns work in the real world [Alexander]:

- **Pattern Analysis (or mining): analyze World → get Patterns from success stories**
    - Successful patterns abstracted from successful projects and pattern instances
- **Pattern Application: apply Patterns → achieve success stories in the world**
    - Successful projects achieved after applying the patterns

Many companies, projects and processes have benefited from the application of these patterns. In my estimation, there are at least 100,000 people doing Enterprise Scrum, and possibly as many as 1,000,000 by a very conservative estimate.

My prediction is that in the future, this style of management will grow in adoption and that it will cause what historians will call perhaps “the Agile Management Revolution”, “the Innovation Revolution”, or maybe “the Knowledge Worker Revolution”. Steve Denning, now a board member at the Scrum Alliance, calls it “Radical Management” [RadicalManagement].

Many of these patterns unfortunately have not been properly documented in the proper organizational patterns form, which follows the seminal work started by Jim Coplien back in 1993 [Coplien]. However, some of us at the ScrumPLOP working group will continue to document these patterns and in the future plan to publish a book on it [ScrumPLOP].

Here is a brief list of some of the early companies and projects where Scrum was first practiced in a generic, scaled or more business-like purposes:

- VMARK – senior management Scrum, 1995
- IDX – first scaled up Scrum, 1996
- Individual – scaled up Scrum, 1996
- CVS/Caremark – scaled-up Scrum (25+ teams), 2001
- New Governance Inc. – company management, 2001
- PatientKeeper – company management, 2001

Today there are many other examples like New Governance Inc., Scrum Inc., Cars.com, SalesForce.com, the Scrum Alliance, Hewitt, ABN Amro, IBM, Total Attorneys, Systematic, Trifork, Spotify, etc. These are the type of companies and the kind of processes, programs and projects in them that led to Enterprise Scrum. Jeff Sutherland gives us very many other example in his new book *Scrum: The Art of Doing Twice the Work in Half the Time* [Sutherland].

There is also a lot of related guidance as to how to structure, manage, and transform organizations that innovate with or without Scrum:

- **good companies** [Collins1], [Collins2], [Hamel1], [Hamel2], [Nonaka-KnowledgeCreating], [Nonaka-PragmaticStrategy], [Nonaka-KnowledgeCreationAndManagement], [Hoshin1], [Hoshin2], [Porter1], [Porter2], [BalancedScorecard], [ToyotaWay], [Nonaka-ManagingFlow], [ScenarioPlanning]
- **good business processes** [Hammer], [Wommack2], [Wommack3]
- **good products and services** [Coplien], [DesignThinking], [CEK], [Poppendieck], [DistributedScrum]
- **agile business management -** [RadicalManagement], [BeyondBudgeting], [LittleBets]
- **business models and startup management -** [LeanStartup], **[**BusinessModelGeneration], [RunningLean], [ProfitZone]
- **teams** [Management3.0], [Atkins], [Katzenbach]
- **culture** [TribalLeadership], [SmartTribes]
- **transformation** [HBR-ChangeManagement], [LeadingChange], [RisingManns]
- **agile and Scrum scaling**, [Eckstein], [DAD], [Schiel], [LeSS1], [LeSS2], [Kniberg], [SAFe], [StoberHansmann], [Cohn3], [Schwaber], [Rawsthorne], [Schliep], [AgileSoftwareDevelopment]
- **agile and Scrum distribution** [DistributedScrum]**innovation** [Peters], [Pink1], [Pink2], [SchwaberSuthterland], [PowerOfScrum], [Stacey]

As we have found, there is a lot of overlap with some of these techniques with companies that practice Enterprise Scrum: they use some of the same patterns.

Customer Driven Scrum Teams – PATTERN 1

The most important Enterprise Scrum organizational pattern is that the we should organize our companies in terms of what is important to our customers as a whole – not by functional departments that become functional silos like Strategy, Marketing, Product Development or Customer Service.

As a general rule:

> create a Scrum Team for every product, service, process, program or project ***that a customer buys or uses***. Enterprise Scrum is a business thing: it's a process framework to deliver the most business value to a customer.

If the market of a company for a product or service is segmented we need different Scrum Teams. If the company is diversified we most certainly need different Scrum Teams, maybe even operating under a different business model.

The main rationale for this is that the timeframes to create a product or service have decreased dramatically due to competition, the product and the process to create need to be more synchronized in time – so we no longer can't create a phase-based or stage-based organization. This means that we can't we use the "business waterfall", strategy->marketing->product development->customer feedback, to create new products and services – it is too slow and unresponsive for the competitive world in which we live today. So must avoid functional departments or component-oriented organizations.

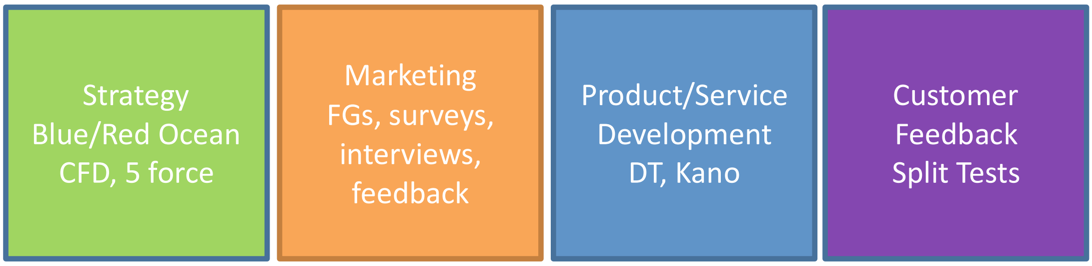


Figure 9. Business Waterfall

Change, feedback and therefore innovation can happen in strategy, marketing, product development, or customer feedback, and we must be willing to inspect and adapt in order to better serve our customers.

Therefore, in Scrum, we prefer all-at-once process models that include strategy, marketing, product development and customer feedback where adjustments can be made on everything as make progress taking advantage of all the feedback from any process area.

Strategy
Marketing
Product/Service Development
Customer Feedback

Figure 10. Enterprise Scrum is ALL at once management: across functions and time

By definition, a Scrum team always has:

- One Business Owner per Product, Service, Process or Program
- One Scrum Master (which could be shared with other teams)
- One Team, which are the people that implement the value list (product backlog), even if they delegate other do actually do the work

- One set of stakeholders, which can be of size zero, or larger
- One value list per Team (even if the list is a sublist of a larger value list (product backlog) for multiple teams.

In a Scrum-managed company where we manage through Scrum everything relevant to our customers (internal or external); it makes sense to talk about our “Scrum portfolio” as being composed of products, services, processes, programs and projects.

Our goal is to create a team that really cares about a certain type of cutomer – a customer-oriented and dedicated Smart Tribe, if you will [SmartTribes]. Both Scrum and Enterprise Scrum have a great deal of cultural overlap with this concept.

Lastly, in a Scrum-managed company, we prefer to have a more flexile budgeting process, rather than a rigid yearly budgeting process that by definition will be always wrong. This matches well with the concept of Beyond Budgeting [BeyondBudgeting].

True Business Value – PATTERN 2

Enterprise Scrum is a process framework to deliver the most business value in the short amount of time. True business value is not “business value” as defined in Scrum, which is mostly “potential business value” through Sprints (1-4 week timeboxes). This is a good start but *True Business Value* in Enterprise Scrum is a business relevant metric: revenue, profit, ROI, market share, or a measure of competitive advantage. The timeframes to achieve true business value vary, but are typically longer than those to achieve “potential business value”.

To have empirical evidence that we have obtained some business value we must measure it appropriately, and then try to improve the delivery of business value in the future. Therefore, a business-oriented Improvement Cycle must be the driver, in terms of measured business value, and our decisions should be made so as to deliver more business value in future Improvement Cycles. Typical business-oriented improvement cycles are of lengths of at least one month for the most agile companies, and quarterly for most other ones.

We will see how we can improve the delivery of *True Business Value* through tools like Lean Startup, Blue Ocean Strategy, Scenario Planning, Profit Zone, etc..

With this expanded view of business value, we see the following benefits:

- **Customer Feedback in any activity of the Business Cycle.** How satisfied with the customers are with our products or services while we develop solutions, deploy them, or while the customers use them. Regular Scrum only manages development – so that we can improve whatever we are *developing* for our customers. But many Scrum practitioners have started to do Deployment and Operations management in their business cycles.
- **Financial Measure.** Find if the operation is profitable or cost effective – which in financial terms is the measurement of business success.
- **Gauge Competition.** How are we doing against competitors or competing

with other alternatives in the market.

- **Improvement or Corrective Actions.** Often product, services or processes are in constant need of improvements of many kinds.

Examples at the business level of these techniques are Lean Startup, Profit Zone, Design Thinking, Scenario Planning, Blue Ocean Strategy and Red Ocean Strategy [LeanStartup], [ProfitZone], [DesignThinking], [ScenarioPlanning], [BlueOcean], [Porter1], [Porter2], [BusinessModelGeneration].

But other insertions business, technical or domain specific are possible.

> ***Enterprise Scrum is a true process framework, with detailed information of where to insert techniques.***

Improving through Iteration – PATTERN 3

A common pattern found in very modern business techniques is that of iteration.

For example, Lean Startup management tells us to implement a business idea and then check and see if it's working in an iterative way [LeanStartup], [RunningLean]. If it is working, persevere; or else pivot.

But the same is true in Beyond Budgeting where we make adjustments as we get feedback from the field to long and medium term plans [BeyondBudgeting]. And the same is true as we iterate to find a working product or service as a Blue Ocean Strategy [BlueOcean]. And we could equally use them to accomplish a different business model [BusinessModelGeneration]. These techniques are of course in agreement with concepts explained in Little Bets [LittleBets].

After an iteration at any one level we could:

> Change our business model, or
> Change our product, service or process – changing the concept of them, or
> Change or add features to an evolving product, service or process, or
> Improve our team, process, or customer service.

Based on maximizing what we defined as business value.

Other techniques like Scenario Planning, Profit Patterns, traditional Strategy, or Kano models come handy to make these decisions.

In Scrum we officially have an iteration called the Sprint, but in Enterprise Scrum we allow the possibility of nesting this iterations in what we call Improvement Cycles. We prefer this name to that of "iteration" because iterations say that you are doing the same. In Enterprise Scrum we don't want to do the same – we want to improve!!! And we want to get feedback though Retrospectives at any level so that we can make any adjustments necessary to improve our products, services, processes, customer satisfaction and our teams.

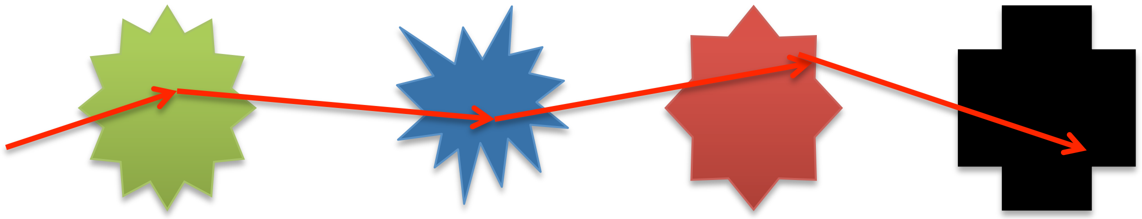


Figure 11. Change: business models, products, services, features until you find the "sweet spot".

Another way to say this is: Enterprise Scrum brings professional iterative and improvement management to any kind of organization, bringing with its all of its cultural tradeoffs of sharing knowledge, cooperation, collaboration and helping each other.

Forecasting and Business Value – PATTERN 4

Another important principle in Scrum and Enterprise Scrum is that *we plan based on what business value measured and accomplished – nothing else, plans, budgets, assumptions.*

Pattern: plan based on measured quantities
Once we define what Business Value is for our team, from a business perspective, there is a need to build expectations (beyond a Sprint) and make forecasts and projections, so that we can plan for contingencies and thresholds and make business decisions. For example, a company may decide that if they offer a new financial compliance management service, it must produce at least 1 million dollar revenue in a year, or else they may decide to do something else.

This is business-level empirical management, which has always been around at least informally, and it is best represented by the Lean Startup techniques as of late [LeanStartup], and which fits other techniques such a Profit Zone [ProfitZone], Blue Ocean Strategy [BlueOcean], etc. We will also parameterize these techniques to optimize Business Value in Enterprise Scrum.

In software, we have developed some techniques using empirical management over the years. See for example Mike Cohn's "Agile Planning and Estimation" [Cohn2].

And we will borrow some of this wisdom to make projections and forecasts through calculations when we get to the Improvement Cycles.

> ***Our forecast is based on previously measured business value, and that re-planning occurs immediately after we measure the business value that we accomplished in past Improvement Cycles– at any level.***

This is more aligned with business needs and concepts like Lean Startup [LeanStartup] and Profit Zone [ProftZone] as discussed above.

When the Improvement Cycles are nested, we can make predictions at a lower level of scale that feed into other higher levels of scale.

For example, our Monthly Sales Improvement Cycles can feed into Quarterly Sales Improvement Cycles, that then can feed into Yearly Sales Improvement Cycles.

Enterprise Scrum Organization

Enterprise Scrum organizations can have more than one team, and these teams can cooperate in different ways not just in traditional style of “functional decomposition” and “command and control” of like traditional organizations.

Let’s try to envision how the “big picture” looks like in Enterprise Scrum in terms of intelligence and organization. Enterprise Scrum is based ***on three intelligence concepts***:

1) cooperating intelligent multi-agents (connectionist intelligence) [MultiAgent],
2) the Unity of Purpose (central delegated planning), and
3) the Subsumption Architecture (subsumed behaviors) [Brooks2].

This is an explanation of what companies that use many instances of Scrum are already doing -- not an invention or wish of mine or of someone else for that matter.

Multi-Agent

Each Scrum Team operates like an intelligent multi-agent inside, powered by the self-organization of the agents that compose it: Business Owner (Product Owner), stakeholders, Team and Scrum Master. The Scrum Team self-organizes to create and update the “master plan” continuously – the Value List (Product Backlog); and then self-organizes to implement it. But also collections of Scrum Teams can form a larger intelligent multi-agent that can self-organize as well. However, and depending on how the Scrum Teams work together, the resulting organization can be either a Multi-Agent organization – with each Scrum Team having their own value list without a “master plan” for all teams, a Unity of Purpose organization – with a “master plan” i.e. a master value list for all Scrum Teams, or a Subsumption-like type organization where the Scrum Teams themselves can represent subsumed behaviors.

Multi-Agents manage themselves by respecting a series of rules of interaction. That lead then to emergent behaviors. For example, the basic rules can emerge into splintering (mitosis-like processes), mentoring, flocking, herding, mimicking, or swarming in general [Swarm].

Unity of Purpose

By Unity of Purpose we mean the hierarchical breakdown of high-level goals from the top. This implies a hierarchy of Scrum Teams working at different knowledge levels at different levels of structure, each with a Business Owner (Product Owner), a Value List (product backlog), a Scrum Master, and an implementation Team, which may be composed of business owners for a lower level of structure and other team members which may be just stakeholders.

For example, upper management desiring a high-level goal, middle management following by starting a project or program, and program or project implementation teams implementing the desired goal. The programs and projects can in turn send feedback back to the parent saying: “we can’t implement this or that”, but leadership in this type of organizations is assumed to come from the top.

The Unity of Purpose architecture provides an overall plan that breaks down details as it gets implemented by different organizational layers and allows people to accomplish a “shared vision” by knowing and focusing on common goals, and finding synergies among their activities to achieve these goals.

Subsumption Architecture

The subsumption architecture [Brooks1], [Brooks2] is based on the concept that several prioritized behaviors can create a special kind of intelligence that is very powerful. It is not based on an overall plan – instead is based on the idea that different behaviors can subsume each other according to some basic rules of interaction. Because there are no models, the slogan for the subsumption architecture is “reality is our model”: there is a high-level goal given by the top-most prioritized behavior but the plans and models to make plans are replaced by a subsumption model. This allows a robot to learn and do very difficult tasks like learning how to walk based on nested feedback loops. And it also allows a company to go “as fast they can towards their goals” but how to get unstuck from blocks quickly through subsumed behaviors.

> ***We need to sometimes radically change the source of our leadership and change our plans because of a finding at a lower level of structure.***

In this case, we may “lead” from any level or layer in the organization: upper management, middle management, program, project or even the technology level.

For example, when research in a pharmaceutical company finds a new discovery that then is passed to middle management so that upper management can switch the company direction based on that. Or when middle management is able to improve a service process through a partnership or logistical improvement and both upper and implementation teams are asked to modify their behavior and plans because of that.

Organizational Choices

Scrum Teams have always been a self-organizing Multi-Agent that have some level of Unity of Purpose through the product backlog but a dose of Subsumption has been a fundamental underpinning of Scrum even for single teams since its very invention by Jeff Sutherland in 1993 [Brooks1], [Brooks2].

Enterprise Scrum formalizes, clarifies and better defines the relationships and communications among the different Scrum instances within the enterprise at different levels of scale and for different business processes to shift the balance among Multi-Agent, Unity of Purpose, or Subsumption type organizations.

In Enterprise Scrum we manage and adapt by constantly listening and communicating through formally defined nested feedback loops among Scrum instances in either: discovery mode, broadcast, multi-cast, or pre-established messaging dependency mode.

Scrum Testing Principles

Umbrella Principle – A larger organization is doing Scrum if the organization as seen from the outside of the organization is doing Scrum – no matter how large the organization may be.

This is important in a scaled Scrum implementation because we want to guarantee that the larger organization is doing Scrum – not just using Scrum for some of its teams underneath. The only way to pass this test is by doing Unity of Purpose Scrum.

Isolation Principle (Inverse Umbrella) – We define a part of a larger organization to be doing Scrum in isolation, if we can prove that the isolated portion is doing Scrum. You can pass this test with either Unity or Purpose of Subsumption Scrum.

These principles allow us to know what parts of the organizations are doing Scrum and how do they cooperate with other parts of the organization.

2. Enterprise Scrum Definition

Enterprise Scrum generalizes and parameterizes the basic Scrum structure and allows extensions so that we can use Scrum for many purposes through configurable parameters. Through these parameters we can instantiate Scrum into useful differentiated forms that are generic, more business-like, and scaled.

Think of this definition as the description of a stem cell in a living organism – with all the cellular proteinomic behaviors, still in a non-differentiated way but with many possible allowed differentiations. In every section of the definition, we also provide some guidelines and examples as to "how to differentiate" Enterprise Scrum instances by using its parameters. It is important to get as good as an instance as we can of Enterprise Scrum and make sure we don't violate any Scrum principles because otherwise we might get a cancerous cell – an incomplete or incorrect differentiation that may lead to problems.

Scrum is already a process framework; but Enterprise Scrum is a much more detailed, general and expandable process framework. The Enterprise Scrum structure at a glance is:

Cultural Values
    Scrum values and BA Elements – honesty, courage, sharing knowledge, cooperation, helping each other, etc.
Value List (was Product Backlog)
    Vision – a concise description of what we are trying to accomplish
    Value List - formed of prioritized VLIs (Value List Items) to express and mange a vision, each adding business value, with a DOR (definition of ready) and a DOD (definition of done), etc.
Reports
    ScrumBoard - one for each for a lower-level Improvement Cycle collection
    Burndowns – one for each Improvement Cycle
    Metrics and Charts – optionally one or more for each Improvement Cycle
Roles
    Scrum Team – composed of the Business Owner, Scrum Master and Team. It could be more than one for scaled up organizations.
    Business Owner – owner of the Value List, ROI, success/failure
    Scrum Master – same as in Scrum – an Enterprise Scrum coach
    Team – as in Scrum, a self-organizing, autonomous, cross functional team
Process
    Vision – creation of the initial vision (written or not)
    Customization - through Enterprise Scrum parameters
    Initial Value List – creating of the initial Value List
    Improvement Cycles - nested for each cycle (e.g. Release, Sprint)
        Planning – plan how much and business value the team Scrum Team will attempt and how they plan to accomplish that
        Sprint Planning part 1 - Agree with Business Owner what to do
        Sprint Planning part 2 - Volunteer for VLIs, create plans for VLIs, and provide detailed estimates for plans
        Execution – work and monitor business value as we achieve it
            Collaborating and sharing knowledge,

        Daily Scrums,
        Updating ScrumBoard,
        Dependency Matrices and
        Updating other metrics and charts
    Review – review and account business value
        Review Value Increment
    Retrospective – find what we are doing well and how we can improve
    Value List Refinement – adjust the vision by managing the Value List
more Improvement Cycles

Product vs. General Business Skin

You may have noticed that we have used “more generic names” in Enterprise Scrum instead of the standard Scrum names. That’s because we are generalizing Scrum to either different levels of scale or other business processes, where there may not be any Products, and where the accepted concept of a Sprints as 1-4 week time-box duration may be insufficient by itself to describe longer-term iterations or nesting. We hope the Enterprise Scrum names might be easier to understand and related by workers of different business processes at different levels of scale. However, these names are just skin-deep – all of the Scrum concepts are the same.

In other words, the world is full of people doing Scrum for things other than “building products” in 1-4 week iterations, and they need a friendly Scrum definition that is applicable to any of their situations. So we need to be extremely careful about our assumptions, the names we use, what is business value and how we can measure it for different situations.

In fact, think of these names as just “a skin” over the traditional Scrum names:

PRODUCT SKIN	BUSINESS SKIN
Scrum	Enterprise Scrum
Product Owner	Business Owner
Scrum Master	Scrum Master
Team	Team
Product Backlog	Value List
PBI (product backlog item)	VLI (value list item)
Sprint	Improvement Cycle – nesting allowed e.g. one week ICs contained in quarterly ICs contained in 1 yr. ICs.
Daily Scrums	Daily Scrums
DOD, DOR	DOD, DOR
Scrum Board	Scrum Board
Velocity	Velocity (measured in varied units – not only effort estimates) e.g. revenue, profit, etc.
Product Increment	Value Increment or <VLI-type1> Increment (where VLI-Type1 is the principal activity)

PSP (potentially shippable product)	UV (usable value)
Sprint Burndown	Burndown
	Metric
	Chart

However, you may also have noticed from the summary above that there are some additional concepts like "metrics" and "charts". These are extensions to regular Scrum that will allow us to measure more than one thing. In Scrum we typically measure velocity in units of effort. But our velocity in Enterprise Scrum is different: it can be the regular velocity in terms of effort, or a different metric i.e. like monthly sales.

Cultural Values

This is one of the most important aspects of Scrum and Enterprise Scrum: the cultural values. And that's why we are starting here. The Enterprise Scrum process executes under the same special culture that Scrum does with the same Scrum values: Commitment, Courage, Respect, Focus and Openness. And jut like regular Scrum the Enterprise Scrum culture is based on the BA elements of sharing: Sharing Knowledge, Collaboration, Cooperation, Asking for Help, and Providing Help to others.

Without these values and BA elements high-order improvements are not possible in either Scrum or Enterprise Scrum. We are trying to build a Scrum-based Smart Tribe [SmartTribes], [TribalLeadership].

Roles

The Scrum roles don't change in Enterprise Scrum we can opt to use different names but we can use the roles at any organizational level and for any business process.

The Scrum Team

The Scrum Team is made up by the Business Owner (Product Owner), the Scrum Master, and the Team, just like in Scrum. Although many of the interactions among the Business Owner, Scrum Master and team are defined by meetings and activities, the Scrum Team itself is cross-functional and self-organizing, just like in Scrum.

scaling

In Enterprise Scrum, there can be one or more interacting Scrum Teams in different modes as we discussed in the introduction: unity of purpose, self-organizing, subsumption, etc.

Business Owner was (Product Owner)

In Scrum, and in Enterprise Scrum the Business Owner (Product Owner) has the same responsibilities:

- provide initial vision with the help of the stakeholders
- create an initial Value List that reflects the vision with the help of the stakeholders
- prioritize the Value List at all times to deliver more Business Value
- manage and redirect the vision throughout the project or process by managing the Value List
- the Business Owner is the only person that can modify or add to the Value List
- Ensuring the Value List is available, transparent and visible to all
- Provide clarifications throughout an Improvement Cycle to the team for the different VLIs (value list items).
- Approve (or not) the results of an Improvement Cycle

Financial responsibilities:
- owns the ROI (return on investment) of the process, product or service.
- owns and controls the budget of the process, product or service.

In some cases, these financial responsibilities, specially in large companies is actually owned by a higher up manager. Beware of this situation: the Business Owner then only owns the vision but is not truly responsible of the success or failure of the process, product or service.

A Business Owner may be in charge of a company, a product portfolio, an individual product or service, or even a feature set of business area.

scaling

In scaled up teams, there might be one or more Business Owners cooperating in different modes: delegation – where there is a Chief Business Owner and helper Business Owners delegated to different business areas; single central Business Owber – where there is a single Business Owner aided by a Business Owner team of stakeholders; cooperating – where different Business Owners from different products, services, or business areas work in cooperation.

Experience indicates that it is best to have a full-time Business Owner. However, experience indicates that if Business Owners have spare cycles they can be can be Business Owners of up to 3 teams.

Team

As in Scrum this is a cross-functional, autonomous, self-organizing team that implements the VLIs (Value List Items) in the Value List within an Improvement Cycle to achieve a Value Increment conforming with a DOD (definition of done). In Enterprise Scrum, the Team may be composed of Product Owners working on a different context. For example, the company management team is composed of the CEO (Business Owner), with each of executives in his Team are Product Owners of Product Portfolios, Service Portfolios, Products, Services, or supporting processes

(HR, Accounting, Finance, etc.).

. The responsibilities for the team are:

- Execute (implement a Value Increment to a Done definition within an Improvement Cycle) using a cooperating, cross-functional, self-organizing team
- Accept (or not) the plans of the Business Owner
- Estimate the VLIs (based on that teams' previous experience).
- Etc.

These are some Team desirable characteristics:

- Full-Time Teams Members (as many as possible)
- Best people in teams
- Self-selected teams – team hires new members
- Each Team manages its own external dependencies
- Balanced Cross-Functional Teams to fulfill DOD (the Team or Scrum Master can help re-balance the team if necessary, the Business Owner funds the new hires) (See parameter **scrumTeam.xFunctionalSkillMatrix)**
- Autonomous Self-Organizing Teams that can work in subsumption relationships with other teams within the organization
- Long-lived teams
- Reward Team Performance
- Adequate Sized teams (3-9 team members is the preferred size boundaries)

scaling

Scaling of Teams is easy: partition the organization is several Scrum Teams that mirror the different customers served. For example, each platform, product, service or business area (if large enough).

Scrum Master

In both Scrum and Enterprise Scrum, the primary responsibility of a Scrum Masters is to be a Scrum coach to the Scrum Team (including the Business Owner and the Team). The Scrum Master responsibilities are:

- Enterprise Scrum coach for the Scrum Team
- Ensure everyone understands and executes Enterprise Scrum
- Owns the Enterprise Scrum process
- Helps the Business Owner and the Stakeholders to configure Enterprise Scrum
- Helping the Business Owner implementing a better Value List to deliver more Business Value through better, concise and well-defined VLIs
- Helping the Business Owner and the Team in making "empirical plans"
- Schedule and provide agendas to al Scrum meetings, enforce agendas, and facilitate all the Scrum meetings
- Remove impediments for the Scrum Team
- Improve the Scrum process in all areas (3 typical areas are: Business Owner interaction, team collaboration and infrastructure)

scaling

Scrum Masters can coach one or more teams and it is better that Scrum Masters expand their responsibilities playing the same role; rather than playing other roles. Experience indicates that the “magic number” is a maximum of 3 teams if the Scrum Master is experienced and the teams are mature. And only one team is the team is new to Scrum (or Enterprise Scrum).

Process

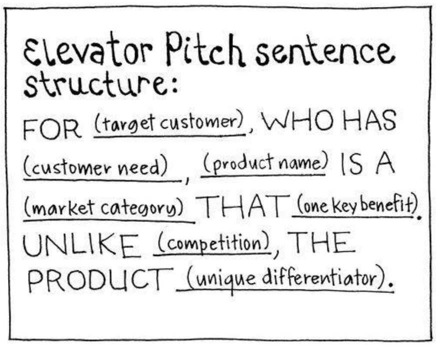
The overall Enterprise Scrum process to define a **valueList** and to accomplish some work through **improvementCycles** is identical to the Scrum in principle; but in addition we must configure the Enterprise Scrum process, and execute the process with the appropriate parameters – even if we have to adjust them on the go.

Vision

Just like in Scrum, we don’t require that a vision is written down, but it is a really good idea to do so. Doing the wrong thing is the biggest risk of any project or process. There are some good techniques to draw a vision such as the Elevator Pitch game [GameStorming]. In the Elevator’s Pitch game we ask the following questions:

- Who is the target customer?
- What is the customer need?
- What is the product, service or process name?
- What is its market category?
- What is its key benefit to the customer?
- Who or what is the competition? Substitutes?
- What is the product, service or process' unique differentiator?

Answering these questions is a good idea even if we don’t write the Elevator Pitch in the format below. However, we can use these answers to write a single sentence that includes all of these answers into an Elevator Pitch:



Elevator Pitch sentence structure:

FOR (target customer), WHO HAS (customer need), (product name) IS A (market category) THAT (one key benefit). UNLIKE (competition), THE PRODUCT (unique differentiator).

Figure 12. Elevator Pitch game

Configuration

The overall process in Enterprise Scrum is very similar than in regular Scrum but it requires at least one extra step: **customization**, because not only do we want to use the Scrum process, we need to customize it for each business process to make sure Scrum is applied correctly. We can configure Enterprise Scrum for company management, marketing, sales, product development, software development, or even things like BPR (business process redesign, SAP implementations, auditing or compliance management. It is a good idea to start with a vision and then create an Initial Value List (Product Backlog), and then start the Improvement Cycles (Sprints) to accomplish our vision. It is always desirable to update the Value List (Product Backlog) as necessary.

We don't have to configure Scrum for every single instance of Scrum, just the first one for a particular business process. For example, once we configure Scrum for Real State Sales, we can use that configuration template and reuse it. However, we must be careful in using templates, and at least review and verify that they properly apply to a new process instance.

> ***Our goal, in time, is to provide a ready-to-go list of configurations for different business processes including the values for the parameters to help any users of Enterprise Scrum customize their processes in an easier and faster way.***

However, this will take some time, and it will evolve with time, of course. Perhaps we can get the global community of Enterprise Scrum users to contribute to an Enterprise Scrum configuration's repository.

The configuration allows us to configure:

- the Scrum Team,
- the Architecture and its management,
- the Business environment,
- the Enterprise Scrum Template we might be using,
- the type of Structure (scaling style),
- the Techniques that we might be using,
- the Value List type and attributes,
- the Value List Instance attributes,
- the Improvement Cycle Type,
- the Improvement Cycle Instance

Here is the list of configurable parameters so that we can customize Scrum for different purposes. Don't be intimidated by this list – it is easier than it looks! Take every parameter and ask the question: what is this for my team? If you are not sure, give your first answer – you might have to discover or change your mind through adjustments what best works for you.

list of sub-parameters = {a, b, c, …}
list of valid values = <a, b, c, etc.>
list of a's = (a)
value of parameter a = [a]

Scrum Team Parameters

Instance parameters describe an instance of Enterprise Scrum at the highest level, providing general information about the identity, purpose, origin, and relationships to other instances.

Parameter. Instance. scrumTeam
scrumTeam – the **scrumTeam** parameter describes who is on the team, **scrumTeam** has several subparameters (the entire list of parameters):

instanceName - **instanceName** is the name of the Scrum instance. A unique identifier that defines what the teams does and is well aligned with the **Principal VLI-type** below. For example, a Scrum instance for company management can be called: "Company Management Scrum Team", "Company Management Process", or simply "Company Management".
scrumTeam.businessOwner – name and contact info of the **businessOwner**
scrumTeam.stakeHolders – name and contact info for each of the stakeholders
scrumTeam.scrumMaster – the name and contact info of the Scrum Master
scrumTeam.team – list of team members: names, contact info
scrumTeam.xFunctionalSkillMatrix – it is useful to track and inventory the overall skills of the team. The overall skills needed are typically known but they can change over time, and therefore, this matrix may be dynamic.

The Vision that created earlier should help us find a good **instanceName,** to recruit the right members for our **scrumTeam (businessOwner, stakeHolders, scrumMaster** and **team),** to better define or understand our **businessCycleType,** and most importantly to define what **businessValue** is**.**

Because Enterprise Scrum teams can be composed of multiple Scrum Teams, the team members at one level can be – but don't have to be, Business Owners for a lower level team. We will see that Scrum instances are related through other parameters as well like Structure Parameters, and the attributes of the **valueListItem.**

Architecture parameters

architectureType – this parameter describes the type of architecture or the team is working on. This reflects which concepts we work on and what relationships may have. The architecture is related to the **domain** and the **instanceName**, but here we describe the kind of abstractions that we deal in that domain from the perspective of what we are building: a system, a building, a business, etc. So the for example, the architecture can be: <"business architecture", "system architecture", "product architecture", etc.>

architectureManagementType – this parameter describes how we evolve the architecture: <self-organizing, Agile Architect, Traditional Architect, External Architecture Team, etc.> Some of these options are more "agile" than others of course. In doing Scrum, we much prefer that the architecture is managed by the entire team.

architect – this coud be “none” or he name of a person that matches the **architecturalManagementType** above. In doing Scrum we much prefer to do self-organizing architectural management but sometimes that is not possible.

Business parameters

domain – this parameter describes the domain in which the Scrum team works. For example, insurance, financial transactions, DNA research, audit, compliance, rocket development, etc.
Parameter. Instance. businessCycleType
businessCycleType -- Development, Customization, Deployment, Business Growth, Business Management i.e. Operations, Revenue Cycle, Phase out, etc.

Sometimes is useful to know what is the high-level purpose of our Improvement Cycles because this might tell us more about what to expect from our Scrum instance.

Parameter. Instance. businessValue
businessValue - this parameter describes what is business value for our Scrum instance, **businessValue** has three subparameters:

description – Scrum is a system to deliver the most business value in the shortest amount of time. Therefore, knowing what business value means to you is one of the most important things when using Scrum. This parameter defines what business value is for our Scrum instance and gives a list of variables to be optimized.
optimizationVariables – the list of business variables to optimize together.
optimizationGoals – describe what type or range of equilibriums we want to achieve

For example, we could be optimizing future sales of a commercial software product; therefore, business value, would be ROI impact of that feature, and we could also include competitive advantage, advancing the state of art, learning something, etc. This maps well to business value in Scrum.

However, in Enterprise Scrum it could be more complex than that. Business Value could be a balance or compromise of two variables like profit and quality.

***

The definition of “true business value” i.e. **businessValue** is very important to determine and configure a template we want to use to create our initial Value List, if the correct **template** is available. (See template parameters below.)

The **optimizationGoals** can link to a **technique** or a **techniqueList**, to optimize our **businessValue**. A **technique** tells us exactly how to complement Scrum to optimize **businessValue** or some other helpful thing through the subparameters: **name, purpose, processStep, howItIsUsed, interfaces, mappings, references** in the **technique**.

Template parameters

Parameter. Business Process. template
template – this parameter describes the template if any that we are using**.** If we are not using a template, this field should be blank or say “none”, **template** has three subparameters:

name – the name of the template we are using
templateType – can be **fromProcessTemplate, fromValueListTemplate,** or **from improvementCycleTemplate**

fromProcessTemplate Specify which Enterprise Scrum Process Template was used, or leave it blank. By convention, the template values would be used but a highlight is used to indicate changes in the values of the parameters.

For example, templates for a specific type of Software Development, or for Company Management, or for Cancer Research may be available.

A database of such templates will be available in the future through participating Enterprise Scrum practitioners.

fromValueListTemplate Specify which Scrum Instance Value List template was used, or leave it blank.

For example, Value Lists to do specific PeopleSoft, Oracle Financials or SAP installations may be available. This is typically done at the project or process level. These templates are called Enterprise Scrum Project/Process Template.

fromImprovementCycleTemplate Specify the Improvement Cycle template used or leave it blank. This is typically used for Scrum management with almost identical Improvement Cycles like Sales Management, or Compliance Management, where there might not be much new in the Improvement Cycle Value List, but of course it is still important to collaborate, share knowledge and improve.

Structure Parameters

These parameters tell us how our Scrum instance is linked to other Scrum instances. We may have some information about this, most dependencies mapped or none at all. However, since Enterprise Scrum is about making Scrum generic, business-like and scalable; these parameters build Scrum into larger organizations by connecting multiple Scrum instances.

Parameter. Structure. parent
parent -- name of the Scrum instance that is the parent, or higher level management for the process.

This indicates that there are one or more VLIs in the Value List for this team that have been delegated by a higher-level team. In other words, there are VLIs that have a **parentVLI.** See VLI attributes in **valueListItem**.

For example, in a product development company, the Portfolio Management Scrum Team oversees all of the products, or even product divisions in the company, so the Portfolio Management Scrum Team is the parent of each of the products, or product divisions.

Typically, a parent aggregates a common metric from the children. For example, in software development, we typically aggregate the velocity of individual but cooperating teams into a large-scale software development effort, as a "Global Velocity". Or in sales, we aggregate in the parent the combined sales of independent teams, products, or services.

Parameter. Structure. contributors
contributors -- names of the Scrum Team instances that contribute to this team's work accomplishment. Notice the difference with team

The **contributors** parameter is a list of all of the Scrum instances that do work for a parent. This means that one or more of our VLIs are the **parentVLI** for VLIs in other (contributor) teams.

The **contributors** typically have a common metric, so that the parent can aggregate it. For example, individual sales teams can have a Revenue or Profit metric that can be aggregated by the parent so that the parent can report on overall performance.

Parameter. Structure. dependsOn
dependsOn – this is the list of all organization we depend on.

Parameter. Structure. dependOnUs
dependOnUs – this is the list of all organization that depend on us

This means that there are one or more VLIs in our team that are dependent on other VLIs from other teams or one or more of their VLIs are dependent on us. We can keep track of these dependencies through the **dependsOnVLIs** parameter in the **valueListItem** parameter. See Value List parameters.

In some cases, these dependencies can be almost static or can change on a daily basis. For example, advertising agencies sometimes produce websites that always depend on compliance and marketing approval, and where these interactions are well-established.

However, in large-scale software development, the relationships of the development teams can vary rapidly – some times daily. Seee also the **dependsOnVLIs** parameter in the Value List **valueListItem** parameter.

It is useful to use a dependency matrix to track these dependencies – specially if they are vary rapidly.

	Team 1	Team 2	Team 3	Team 4

Team 1	N/A	X		X
Team 2	X	N/A		
Team 3	X		N/A	
Team 4			X	N/A

***

In Enterprise Scrum we can accomplish the work with one or more teams. When we start a project it is easier to start with one team and then add more teams as needed. However, once you master scaling, it is possible to start with multiple teams at once but it is not the recommended way to get started.

A **parent** Scrum Team can add one or more Scrum Team as **contributors**. It is also convenient to keep track of which teams we **dependsOn** – the list of teams we depend on, and who **dependsOnUs** – a list to track which teams depend on us. These relationships can be almost static or change rapidly in time, it really depends on what we are doing.

Recall from the introduction that our number one organizational pattern is the Scrum Team.

Scrum Team

Pattern: Scrum Team

We typically form new helper Scrum Teams around business areas because then there is a one-to-one correspondence among:

Customer option ->
    Business Area ->
        VLI-type ->
            Subsystem (if it a by product exists) ->
                Scrum Team

For example, in a benefits management system we would strongly suggest forming teams for Pension, Defined Contribution (401K), and Health and Welfare plans. It also work on other domains, for example, real state sales can be split in Retail, Mansion, Vacation or Commercial. Larman and Vodde call these "feature teams" – a good name for software or product development [LeSS1], [LeSS2].

Be aware that these relationships may change over time, so we could rearrange the team's structure as needed.

Add a parent Scrum Team

Pattern: Add a Head (Add a Parent)

Suppose that the opposite happened teams were added as needed to offer more services to the customer but now no controls the overall picture. When this is the case, it is a good idea to add a **parent** Scrum Team to coordinate the "face to the customer". The Business Owners of each of the Scrum Teams may form themselves a **parent** Scrum Team that has a ***constrained autonomy*** since it depends on its contributors.

Unity of Purpose

Pattern: Unity of Purpose
The relationship of parent to contributors may vary. In some cases is more delegation: **contributors** work for the **parent** with feedback from the trenches as to why they can do but trying to achieve a “grand vision” from a Chief Product Owner. This is needed sometimes in some environments but there are other options.

Subsumption

Pattern: Subsumption
It is also possible to operate through a set of subsumed behaviors: the **contributors** just contribute to the success of the **parent’s** main behavior, choosing the best behavior among the Scrum Teams through a defined set of behaviors that subsume each other. The **parent** may give a direction to the **contributors** but then as the **contributors** teams find market information, they may send direction (information), that affects the **parents** overall direction.

Multi-Agent

Pattern: Multi-Agent
It is also possible to have a softer cooperating self-organizing relationship: the **contributors** just contribute to their success self-organizing to achieve their individual goals, and resolving conflicts and dependencies as needed with other Scrum Teams – there is no **parent** or there is only a very high-level **parent**. Every team leads itself communicating, co-adapting and flocking with other teams.

As a general rule is that the **contributors**, should operate as autonomously as possible. This is a critical pattern in Scrum-managed organizations: direction or new information may come from any other layer, and therefore the entire organization needs to quickly adapt to this new direction generated by the new information.

Techniques Parameters

These parameters tell us about what techniques we are inserting into the Scrum framework to specialize our Scrum instances for whatever purposes we need.

Parameter. Techniques. technique
technique - technique describes a technique that has been inserted into our Enterprise Scrum instance to help us in a particular way. The subparameters to describe our technique is:

name – describes what the technique does
purpose – what purpose does the technique have
processStep – the exact process step where the technique is introduced
howItIsUsed – a description of how the technique is used in our Scrum instance
interfaces - what inputs do the technique need from Scrum, what outputs to Scrum does it produce
mappingsToScrum –and how exactly the outcomes are mapped to Scrum and its artifacts, for example, the Value List
references – provides where we can find more information about it

For example, we may introduce a Blue Ocean strategy's Strategy Canvas as business technique to help us find a better company, portfolio or product direction. Let's assume that is a product.

Therefore our Strategy Canvas **technique** is configured:

name – Strategy Canvas
purpose – identify strategic factor values so that we can better create and prioritize our Value List in order to position our product in a unique niche
processStep – the technique should be used before the creation of Value Lists – both initially and after every Refinement
howItIsUsed – the product factors are initially analyzed and then re-analyzed at the beginning or every Refinement so that we can create a better Value List
interfaces – the Value List can be used as input to create the current state of the Strategy Canvas. After analysis is market, competitors, suppliers, and other factors; modifications can be made to the Strategy Canvas, and its output can feed into the Value List.
mappingsToScrum –the outcome of the Strategy Canvas helps us as high level guidance for the creation or modification (reprioritization) or our Value List. Typically a Strategy Canvas factor maps into multiple Value List items (VLIs).
references – Blue Ocean Strategy books, websites, internal documents, previous analysis documents, etc.

Other techniques from Lean Startup, Profit Zone, Design Thinking, Scenario Planning, Blue Ocean Strategy, Red Ocean Strategy can be inserted that way as well.

Parameter. Techniques. techniqueList
techniqueList -- techniqueList is simply a list of all the techniques used within our Scrum instance.

Value List Parameters

The Value List (or Backlog), is formed by a list of VLIs (value list items or PBIs, product backlog items). Each VLI (PBI), is supposed to add a granule or quantum of business value by definition, or have another VLI require a particular VLI to add business value.

Parameter. Value List. principalVLIType
principalVLIType == [VLI-type 1]

For example, the dominant VLIs for RS sales are: Real-State property sales, because this is our principal activity in Real-State sales. In software development, the principal type of work is the deployment of functional requirements, because that's how we primarily deliver Business Value to our users. We should at least know what is our principal activity, but sometimes is fuzzier, so we may want to get the values for all the VLI types before (the next parameter in the list).

Parameter. Value List. VLITypes
VLITypes – **VLITypes** is the list of VLI types that the Scrum instance supports

These are the different types of work that we are planning to do with our instance of Scrum through Improvement Cycles and deliver them as Business Value. For example, in an instance for software development, we could do infrastructure VLIs, architecture components VLIs, implementation of functional requirements VLIs, etc.

Enterprise Scrum can handle any type of mix of work, but we prefer to concentrate on a specific kind of work. We usually refer to this as the “color mix”, or “colorization” of the Value List (Backlog) i.e. the types of allowed VLIs.

Examples of VLIs:

Business Process	VLIs
Software Development	Software Requirements, Infrastructure (Testing, Integration, etc.), Technical Components.
Hardware Development	Hardware Features, Acquisition, Infrastructure (Testing, Integration), functional requirements
Compliance Management	Deployed Business Processes with new Compliance Standards, Compliance Documents, Policies and Procedures, Internal Control definitions, Board-level reports, etc.
Sales	Net Profits of: Properties, Automobiles, retail goods, etc., Partnerships, etc.
Strategy	Strategic Key Initiatives, implementation
Customer Service	Customer Service Standards, customer service processes, customer satisfaction, etc.
	Etc.

Parameter. Value List Item (type). prioritizable.
prioritizable Yes or No.

There are some situations where the Value List may not be possible or desirable to prioritize in terms of business value. For example, in a sales process, some real state properties may not have a higher priority than some other ones, or we may choose not to give them a priority. The same is true for a new car sales process or to a customer service process.

Parameter. Value List Item (type). orderingTechnique.
orderingTechnique - Name and description of the technique that we use to order by priority our Value List.

In Scrum the default prioritization technique is a comparison in the relative Business Value to be delivered by each PBI.

However, in Enterprise Scrum we have specified what business value is and the variables to optimize in our Scrum instance. Therefore, these **orderingTechnique** tells us exactly how to use this variables in our Value List to order by priority.

Parameter. Value List Item (type). sizingUnit
sizing unit - specify what units to use to size

For example, a Sales process may size its backlog by revenue or profit not by effort.

Parameter. Value List Item (type). sizeUncertainty.
size uncertainty == estimated average percentage of uncertainly in a VLI type.

Every VLI is uncertain but some VLIs are more uncertain than others. If we can determine where most uncertainty concentrates among our VLIs, we can plan more carefully.

For example, in research and development, if we chose “effort” in some unit of time as our sizing unit, we may determine that our effort estimates could have a standard deviation of 100%. Ideally, we would measure this uncertainty from previous experience, but this uncertainty could be estimated. In the case where we really don’t have any idea of the uncertainty to be expected we could leave this value as UNDETERMINED, and then measure the uncertainty later as work is performed.

Parameter. VLI Item (Type). dependentVLIs
dependentVLIs - Yes or No

Value List Items are sometimes dependent on each other. Therefore we must specify if the components are dependent or independent for each VLI or VLI type.

For example, if we are monitoring internal controls and each control represents a VLI, the controls are typically independent from each other. In hardware development, there are typically many dependencies among components, but some components do not interact with each other.

The Daily Scrums are still very useful even when we have independent VLIs, because team members can still help each other, collaborating, picking up work from other team members, and exchanging knowledge and experience.

Parameter. Value List type. DOR

DOR - this is the DOR standard for a VLI type.

We should provide a DOR for every VLI type if possible, because this determines a standard by which all VLIs of that type are ready to be worked on.

Parameter. Value List type. DOD

DOD - list is the DOD standard for a VLI type.

Each VLI type should have a DOD if possible because this determines the standard by which all VLIs of that type are done.

Sometimes, we can assign DODs to VLI types. This is useful. But we need to be careful – each VLI must satisfy its own DOD which may have to be unique.

Parameter. VLI (type). VLIsToDeliverableMap.

VLIsToDeliverableMap -- define what type of components will the VLI types may result into.

To be clear, we make a distinction between the goal – VLIs, and the thing that was delivered to fulfill the goal Increment Component(s).

For example, in software development our goal is to develop a feature for a user; but we deliver an Increment Component – the software component of the Increment that fulfills that goal. Therefore, the Deliverable Map is a table that maps the functional requirement VLIs to the components in the Product Increment that deliver these requirements.

In software development, we are so used to deliver either a Product Increment, or a set of components that sometimes we forget this mapping could be important.

Let's give another example -- a financial company that needs to be compliant with the Investment Company's Act. Its goals for a Compliance Improvement Cycle, may be to be compliant with some or all of the appropriate SEC's statues in the Investment Company Act.

As part of this goal, it writes policies and procedures, defines and executes internal controls to prove that it is compliant with the policies and procedures, and provides activity, testing and other compliance reports.

The VLI is: Monthly Compliance with the SEC Investment Company Act Iteration
The Increment Components to achieve this goal together with its acceptance criteria are:

- **the compliant processes** in the organization as measured by the set of internal controls, which represent its acceptance criteria
- **each of the internal controls** with its acceptance criteria
- **the internal document titled "Investment Company Act Policies and Procedures"** that outlines the compliance policies and procedures to be enacted by the operations' processes, which map to the internal controls

described above. The acceptance criteria of the internal documents is that they allow the organization to be compliant with the Investment Company Act.
- **the Internal Controls Monthly report**, with the acceptance criteria of having passed a certain percentage or list of mandatory regulatory controls.
- etc.

Value List Instance Parameters

Parameter. Value List Item (Type). valueList(valueListItem)
valueList(valueListItem) -- list of all VLIs

In Scrum this is called the Product Backlog but because we build products, but in Enterprise Scrum we are interested in delivering the most business value as possible.

Parameter. Value List Item (Type). valueListItem.
valueListItem -- defines a value list item – a granule or quantum of business value. It can be a direct contribution or an indirect contribution of business value e.g. an integration server to be able to deliver functions in software development, or software to support a sales process, etc.

The **valueListItem** has a mandatory list of VLI attributes as well as optional ones:
name – name of the VLI
description – description of the VLI
priority – priority of the VLI (typically a number 1-N) if it is **prioritizable**
size – size of the VLI in the **sizingUnits** chosen
DOR – definition of ready. It may be inherited for each VLI from the VLIType
It's a list that we call **entryCriteria** (to match the DOD "acceptance criteria").
DOD – definition of done for the VLI. It may be inherited from VLItype
We call the elements of this list **acceptanceCriteria**.

Notice, this is a "definition of done", not of "partially done". The different condition within a DOD must specify when there is "nothing else left to do".

In strategy, the definition of done for a VLI that has the description "Working with a partner in China", may have a DOD that includes:

- Signing the partnership contract
- Having at least 3 customers through that partner in China by end of Q1 2014

That is, when we have achieved these things, an only then, we would consider our VLI "done".
planForVLI -- a plan for VLI or a VLI type**.** A plan can be a list of tasks, a list of components, a list of sub-goals, workflow or state machine, typically annotated with owners, and may include an estimated time to completion or effort (or date to be finished). The plan needs to satisfy the DOD for the particular Scrum instance.
parentVLI – the parent VLI if there is one
dependsOnVLIs – the list of dependent VLIs if there are any
selectable – yes, if the VLI is selectable into an improvement cycle, no otherwise

volunteers – the list of volunteers for a VLI
Created by,
Added in Date,
Budget,
Revenue,
Profit,
Etc.

However, in Enterprise Scrum, because we typically do work with multiple teams, need the **parentVLI** parameter, so that larger scale VLIs can be traced to lower level VLIs and vice-versa. When there is one or more VLIs that have a common **parentVLI**, we summarize that in our instance of Scrum saying it is the **parent –** see above in Structure Parameters.

Additionally, it can prove useful, to have a list of VLIs that each of the VLIs depend on, if this list is more or less static. However, if these relationships change fast, it could be hard to track them. When there is one of more VLIs in our Scrum instance that depends on the VLIs from another Scrum instance, we say that our Scrum instance **dependsOnVLIs** that instance. See above in Structure Parameters.

Other attributes can provide different metrics that may make sense in our situation. We will describe and give examples of these metrics later when we talk about Improvement Cycles (Sprints). Some of those metrics may define alternate velocities. For example, they may define target profits in a sales process per Improvement Cycle, etc.

Improvement Cycle Parameters

Parameter. Improvement Cycle. listOfImprovementCycles
listOfImprovementCycles -- specify the list of improvement cycles.

A Scrum Team can have one of more Improvement Cycles. For example, a development team may have 3 Improvement Cycles:

2 week Development Improvement Cycles
3 months Release Improvement Cycles (6 Development Improvement Cycles)
1 year Business Validation Improvement Cycles (4 Release Improvement Cycles, 24 Development Improvement Cycles and some time off)

It is a good idea that all the **contributors** of a Scrum Team automatically inherit all of their Improvement Cycles of the **parent**.

Parameter. Improvement Cycle. improvementCycle
improvementCycle – defines an improvement cycle type: how we are planning, accomplishing, measuring, and what kind of work we are improving.

The subparameters of an Improvement Cycle are:
name,
level,

length,
metrics (metric(name, description, unit, frequency}),
charts(metricChart {name, description, metric, type}),
scrumBoard,
dependencyMatrix,
valueIncrementName,
usableValue,
calculations (calculation {description, formula})

For each of the Improvement Cycles listed above we need the following parameters:

name -- This is the name for our improvement cycle. We typically use the name of our Work Type == <VLI-type 1> and a frequency. For example, "Weekly Sales Improvement Cycle".

length – it is the length of the Improvement Cycle. First layer Improvement Cycles typically are 1-4 weeks of duration. Second layer vary from 1 month through a year. Third layer are typically at least six month long and are the business validation level.

metrics -- list of metrics to be used in the Improvement Cycle.

In regular Scrum we typically just have one metric velocity, always measured in units of effort, so that what we tend to optimize: velocity. However, in Enterprise Scrum we are allowed to have multiple metrics or measurements. It is convenient to know all the metrics that we are using so that we can balance the optimization of our Improvement Cycle accordingly.

For each metric we need to define a **metric**.

metric(name, description, unit, frequency) - define a generalized measurement to track progress or status in an Improvement Cycle. Typically an **optimizationVariable** is used in a cumulative or non-cumulative way, either as either growing towards a goal, or as the "left to the goal".

It is important to choose relevant business-value like metrics that are important to the customer – not just arbitrary things that are convenient to measure. For example, in a sales process, rather than measuring an internally important measurement such as "effort per sales", it is better to measure "customer satisfaction".

This allows Enterprise Scrum to define one or more useful metrics, as opposed to just have the canonical "velocity" measured in units of effort. Metrics have the purpose to measure how close we are to our goals.

For example, some metrics could be revenue, profit or customer satisfaction in an Improvement Cycle.

Each metric has the following attributes:

name – the name of the metric. For example, "net profit"

description – tells us what it measures and how to measure it
unit – it tells us the units we are using to measure the metric
frequency – suggested frequency of measurement

charts – this is just the list of charts for the Improvement Cycle

metricChart (name, description, metric, type) – this parameter defines a chart where the metric is used.

The attributes of a chart are:

name – this is the name of the chart
description – it tells you what it is and how to build it
metric – it tells us what metric to use (see description above)
type – burnup, burndown, other

Typically, we graph the metrics on the Y-axis and time on the X-axis. The resulting graphs can be both burndowns or burnups, or just a graph of the metrics like "average daily customer satisfaction" when customers are taking surveys after a service appointment, for example.

scrumBoard -- specify which type of Scrum Board we will use to manage the Improvement Cycle workflow

Having a DOD (definition of done) allows us to workflow our VLIs across the Scrum Board in the same way that we do in Scrum:

- Not selected – VLIs that have not been selected into an Improvement Cycle.
- Selected – VLIs have been selected into an Improvement Cycle. In the case VLIs are not selectable, all VLIs are assumed to be selected (see the Selectable parameter below), or in WIP if there is some work done on them already.
- WIP (work in progress), for VLIs that have been selected for this Improvement Cycle and that some work has been done on them. The WIP column can be furthered customized to match a specific workflow, or leave it as "free flow" until the DOD is achieved.
- DONE – for VLIs that satisfy the DOD – hopefully with nothing else left to be done and the product owner, or someone delegated from the product owner has approved of the VLI.

This is the same workflow as defined with Scrum but just with a couple of variations as to what can be given order or priority, what VLIs can be selectable, and respective DOR and DODs.

dependencyMatrix – is a chart to map dependencies for an improvement cycle of either VLIs, persona or teams. This defines the format to be used by the Improvement Cycle instances.

valueIncrementName – This is the name of our increment. We typically call the

"VLI-type1 Increment". For example, "Profit Increment" or "Sales Increment" or "Product Increment", etc.

usableValue -- define what is usable value.

In regular Scrum this Business Value comes as Product Increment that has the quality to be Potentially Shippable Product. Potentially Shippable Product means that this is a product that has nothing left to be done to be shipped. However, when doing Enterprise Scrum to manage other business processes we may not necessarily be "shipping a product". Moreover, this is only "potential value".

In Enterprise Scrum, we are more interested in tracking the **usableValue** if possible – not the "potential value".

calculation (description, formula) – a **calculation** is a derived quantity that is useful to know for an Improvement Cycle

calculations (calculation) – calculation is the list of calculations that are useful for an Improvement Cycle. For example, budget, schedule, customer satisfaction, compliance level, etc.

Improvement Cycle Instance Parameters

Parameter. Improvement Cycle Instances.
improvementCycleInstances(improvementCycleInstance)
improvmeentCycleInstances – this is the list of Improvement Cycle Instances saved with all of their data

Parameter. Improvement Cycle. improvementCycleInstance

For each instance of an Improvement Cycle we capture the following information:

improvementCycleInstance (
icID,
initialValueListForImprovementCycle(VLI),
doneValueListForImprovementCycle(VLI))
retrospective(well, improvements),
metricInstances (metrics),
chartInstances (chartInstance),
calculationInstances,
scrumBoardInstance,
dependencyMatrixInstance,

icID – this are the name or value identifying a particular Improvement Cycle. For example, "Sprint 23, Release 2.1" in a software development.
initialValueListForImprovementCycle(VLI) – this is the initial **valueList**
doneValueListForImprovementCycle(VLI) – this is the **valueList** that got done

(passed the DOD and was approved by the Business Owner).
retrospective(well, improvements) – this captures the results of the retrospective. It is desirable to bring these results to the next retrospective as a starting point.
metricInstances(metrics) – these are the instances of metrics generated with a particular frequency
chartInstances (chartInstance) – these are the charts for that particular Improvement Cycle instance
calculationsInstances – this is the list of calculations for the Improvement Cycle
ScrumBoardInstance - this is the instance of the ScrumBoard used
dependencyMatrixInstance – this is the ongoing dependency matrix tracking internal and external dependencies

list of sub-parameters = {a, b, c, …}
list of valid values = <a, b, c, etc.>
list of a's = (a)
value of parameter a = [a]

ALL Parameters short list

Scrum Team Parameters
scrumTeam {instanceName, businessOwner, stakeHolders, scrumMaster, team, scrumTeam.xFunctionalSkillMatrix}

Architecture Parameters
architectureType
architectureManagementType
architect

Business Parameters
domain
businessCycleType
businessValue {description, optimizationVariables, optimizationGoals}

Template Parameters
template {name, type=<fromProcessTemplate, fromValueListTemplate, fromImprovementCycleTemplate>}

Structure Parameters
parent
contributors(instanceName)
dependsOn
dependOnUs

Techniques Parameters
technique {name, **purpose**, **processStep**, **howItIsUsed**, **interfaces**, **mappings**, **references**}
techniqueList(technique)

Value List Parameters
principalVLIType = [VLI-type1]
VLItypes (VLItype)
prioritizable
orderingTechnique
sizingUnit
sizeUncertainty
dependentVLIs
DOR (entryCriteria)
DOD (acceptanceCriteria)
VLIsToDeliverableMap

Value List Instance Parameters
valueList(valueListItem)

valueListItem {name, description, priority, size, DOR, DOD, planForVLI, parentVLI, dependsOnVLIs, selectable, volunteers, Created by, Added in Date, Budget, Revenue, Profit, Etc.}

Improvement Cycle Parameters
listOfImprovementCycles(improvementCycle)
improvementCycle {name, level, length, metrics(metric{name, description, unit, frequency}), charts(metricChart {name, description, metric, type}), scrumBoard, dependencyMatrix, valueIncrementName, usableValue, calculations(calculation {description, formula})}
improvementCycleInstaces(improvementCycleInstance)

Improvement Cycle Instance Parameters
improvementCycleInstances(improvementCycleInstance)

improvementCycleInstance (
icID,
initialValueListForImprovementCycle(valueListItem),
doneValueListForImprovementCycle(valueListItem),
retrospective(well, improvements),
metricInstances,
chartInstances,
calculationsInstances
scrumBoardInstance,
dependencyMatrixInstance
)

Initial Value List

In Enterprise Scrum we prefer to use the name Value List at every level of scale or for different business processes, because the name “Product Backlog” in Scrum doesn’t reflect well situations when we are not working on products.

> ***Enterprise Scrum is a generic management framework based on Scrum. Our goal is to produce business value – of any kind, not just business value from “products”.***

However, just like we do in Scrum, we always try work on things that deliver most business value first. We defined what **businessValue** is for our **scrumInstance** in the configuration and what type of activity it is through the **businessCycleType**.
In Enterprise Scrum we use the **valueList** to keep the list of **valueListItems.** (compare to Product Backlog items). As in Scrum, this is a list of items to deliver granular **businessValue** through each VLI. In Scrum it was required that the Product Backlog items were defined, prioritized (ordered by business value) and sized (in effort to certain DOD (Definition of Done)) before we could do any work. And it was implied from these requirements a default DOR (Definition of Ready): to be at least defined, ordered by priority and sized (in effort).

In Enterprise Scrum we need to be much more generic than that because we can use Scrum to manage any business process.

Template Parameters

One possibility is that we could be using an Enterprise Scrum **template** defined before. Templates can come in three flavors:

1) templates for processes (Sales, etc.): **fromProcessTemplate**
2) templates for projects or other Value Lists: **fromValueListTemplate,** or
3) templates for Improvement Cycles: **fromImprovementCycleTemplate.**

It is assumed that these templates are somewhat repeatable choices but not necessarily identical: we can take a template and then add or change it to fit our needs. For example, at New Governance Inc., we have used value list templates to install our compliance management software for different clients. And we have heard of others doing that for SAP or Peoplesoft installations.

New Project, Process, Product or Service

However, we can also create the **valueList** for a brand new process/project with its own parameters for the first time. Ideally, we would have configured these parameters correctly to begin with, but we are also allowed to modify the parameters as we find more information about the process/project. Think of the initial parameters defined in the configuration just as a starting point that can change as we find more about what we really want to do or even if the nature of the process changes. In other words, we follow the same agile iterative ideas and concepts to configure Enterprise Scrum:

> ***we discover the nature of your process and create or change configurations as you go, testing that they work along the way.***

The **principalVLIType** tells us the main purpose of our project or process. For example, it could be Sales or Compliance Management, or functions of a software system, like User Stories [Cohn1]. The VLI-types tell us about all the different types of work that we can do. For example, it could include setting up infrastructure for software development projects, or compliance management, or company management, etc.

Sometimes the Value list can be prioritized but sometimes it is not. We would know this from the **prioritizable** parameter. But to order the list by priority, we have to state specifically what is the **orderingTechnique** to be used.

In order to provide a size, we must first decide what is our **sizingUnit**. Remember in Enterprise Scrum we don't have to size by effort, we could size by many other metrics: revenue, profit, customer satisfaction (accumulated), number of controls, features, effort (in some time or relative unit), etc. Since this parameter is our main sizing unit, this parameter will also determine how we are going to measure velocity. Later, we will see that we will be able to have multiple **metrics**; therefore, it is possible that we may have multiple “velocities” or measurements, and that we may choose to optimize one of them, or seek a balance among them. For example, in a sales process, we may monitor both profit and customer satisfaction, and optimize for a desired combined level of profit and customer satisfaction.

It is nice to have some idea about the **sizeUncertainty**, but for new projects or new project types in an organization it may be impossible to know this value.

Typically, we also have a very good idea whether the VLIs are dependent or independent. We used the **dependentVLIs** parameter to note that. This will determine the nature of our Daily Scrums because if they are dependent, we have to monitor the dependencies. If they are not dependent, the Daily Scrums will serve more as an interchange for knowledge, collaboration and shifting responsibilities.

Every Enterprise Scrum Value List item (or type) needs to have a **DOR** (definition of ready), and a **DOD** (a definition of done). The **DOR** tells us when we can start working on a VLI. And the **DOD** tells us when we are done – what are all the conditions to call something done.

Even though the **DOD** tells us when we are “done”, sometimes is nice to have a map that explicitly tells us what we should expect to come out exactly from a VLI. We call this map the **VLIsToDeliverableMap**. For example, in compliance management, we may choose to have a VLI as showing that a control has passed the test(s) to be valid, but there might be a document, a spreadsheet, or database entries reflecting this – it tells us about the exact format that we expect the VLI to be delivered, not only the conditions that it needs to pass to be done; although the **DOD** may very well refer to these deliverables.

Because Enterprise Scrum is a generic management process, we may need attributes or other **metrics** in a **valueListItem** to track progress. The **valueListItem** attributes could be things like estimated cost, board approval, etc.; or **metrics** like quality level, revenue, profit, etc. Each VLI should always have:

name – name of the VLI
description – description of the VLI
priority – priority of the VLI (typically a number 1-N) if it is **prioritizable**
size – size of the VLI in the **sizingUnits** chosen
DOR – definition of ready. It may be inherited for each VLI from the VLIType
It’s a list that we call **entryCriteria** (to match the DOD “acceptance criteria”).
DOD – definition of done for the VLI. It may be inherited from VLItype
We call the elements of this list **acceptanceCriteria**.

Also, because we expect to connect several Scrum instances in either delegating or cooperating modes, the **valueListItem** should always include the **parentVLI** and **dependsOnVLIs** parameters, because they will link VLIs from different Scrum instances.

The **valueListItem** attributes include extensions over regular Scrum that allow us to track more than just the one variable we typically track in Scrum: **size (in effort units)**. These attributes and measurements will allow us to customize Enterprise Scrum for other types of projects. There is typically a connection from

valueListItem with a **metric** in the list of **metrics**. For example, we may track profit for a sales operation as a **metric** but this may not be an attribute of the "properties for sale" VLI type in the Value List.

Scaling

When the Value List is very large, or has different complex VLI types, we may want to divide the work among several Scrum Teams. As said before, we typically choose business areas for partitioning a broad value list. For example, if we are building a large complex software development system such as a benefits management system that includes pension, 401K (saving plan), and health and welfare plans; we may want to partition the value list along those business areas. This is the so-called extended Conway's Law that says that there is a one-to-one correspondence among:

Business Areas -> VLI-type -> subsystem -> Development Teams

This also applies to other domains. For example, real-state sales, where we may want to partition the different sales areas corresponding to different customer types such as retail, mansion, vacation and commercial areas:

Commercial Customer -> Commercial Properties -> Commercial Sales Team

Improvement Cycle Concepts

Once we have an initial **valueList**, our Enterprise Scrum team can start accomplishing some **businessValue** through **improvementCycles**. In Scrum the only improvement cycle is a Sprint: a 1-4 week time-box where we can accomplish some Business Value (or potential business value). However, even though multiple improvement cycles are technically not part of Scrum, people that use Scrum for a variety of purposes around the world know the second level improvement cycles as Releases.

Nesting Improvement Cycles

In Enterprise Scrum a single team can have a nested **listOfImprovementCycles** of diverse lengths that can be used for different purposes. These improvement cycles need to have a **name** and a specific **level** and **length**, for example, "Quarterly Improvement Cycle". The **level** is the level of nesting from a day. For example, in Scrum, the Sprint is **level** 1, and the Release is **level** 2.

The different improvement cycles can also define what we are measuring as progress and **businessValue** in each one of them. It is convenient to choose compatible metrics in nesting, so that the lower-level improvement cycles can feed or build the higher-level improvement cycles. For example, a sales team that defines their velocity as "net profit" in each of its 3 nested Improvement Cycles:

1 week "Weekly Sales Improvement Cycles"

3 months “Quarterly Sales Improvement Cycles”
1 year “Yearly Sales Improvement Cycles”

Therefore, there will be 52 “Weekly Sales Improvement Cycles Instances”, 4 “Quarterly Sales Improvement Cycles”, and 1 “Yearly Improvement Cycle” in a year. This sales team may also monitor product quality and customer satisfaction as well. Every Improvement Cycle has a PE3R structure (Planning, Execution, Review, Retrospective and Refinement), at any level and for any purpose.

However, to make predictions, the Scrum Team must re-calibrate its plans for higher improvement cycles with every lower level **improvementCycle** that it finishes through our velocity. For example, we can make a Quarterly Sales Forecasts based on our Weekly Sales Improvement Cycles velocity, but we have to recalibrate our plan at the end of every Weekly Sales Improvement Cycle.

Sometimes we can combine the PE3R meetings. For example, in Scrum sometimes we can make the last Sprint Review also be the Release Review. And we sometimes combine the Retrospective and Refinement as well. We can do this to save time as long as the **improvementCycles** have a compatible **metric**.

Type vs. Instance

In Enterprise Scrum we make the distinction between an **improvementCycle**, and an **improvementCycleInstance**. The **improvementCycle** is where we define how the **improvementCycle** works, and the **improvementCycleInstance** is the records that results from execution an **improvementCycle**.

Each improvement cycle has a number of attributes that define it. In the example above we had a one-week improvement cycle named “Weekly Sales Improvement Cycle”. We can define the rest of the attributes as:

name – ”Weekly Sales Improvement Cycles”
level - level 1, since we are counting the nesting from the day frequency
length – 1 week
metrics(metric{name, description, unit, frequency}) – net profit, product quality, customer satisfaction
charts(metricChart{name, description, metric, type}) – e.g. for profit: metricChart(“profitBurnup”, cumulative profit in a week, net profit, burnup)
scrumBoard – standard without value list for improvement cycle since items are not selectable
dependencyMatrix – none, assuming sales are independent
selectable – no, any items can be sold during the improvement cycle
valueIncrementName – retail sales
usableValue – profit
calculations(calculation {description, formula}) – quarterly sales projection

Notice this information is only defining how the “Weekly Sales Improvement Cycle” works – it is not capturing any data for any particular improvement cycle.

However, each improvement cycle will also be instances:
improvementCycleInstaces(improvementCycleInstance)
that will record specific data in each one of them.

Each **improvementCycleInstance** has a number of additional attributes that are important. To continue our "Weekly Sales Improvement Cycle" example:

- **icID –** it can be improvement cycle 2 or the 2nd Quarter
- **initialValueListForImprovementCycle(VLI) –** none, because it is not selectable
- **doneValueListForImprovementCycle(VLI) –** this is the list of retail items that got sold
- **retrospective(well, improvements) –** the list of what the store doing well to sell, and the list of improvements after the improvement cycle 2 or the 2nd Quarter
- **metricInstances(metrics) –** the actual daily numbers of net profit, product quality and customer satisfaction for the improvement cycle 2 or the 2nd Quarter
- **chartInstances (chartInstance) –** the charts corresponding to the actual numbers in improvement cycle 2 or the 2nd Quarter
- **calculationsInstances –** this is the list of calculations for improvement cycle 2 or the 2nd Quarter
- **ScrumBoardInstance -** this is the instance of the ScrumBoard used for improvement cycle 2 or the 2nd Quarter
- **dependencyMatrixInstance –** none since retail sales are independent

Value Increment

Our goal in Enterprise Scrum is to deliver true **businessValue** and therefore to deliver a value increment that has business value. We give a name to this value increment: **valueIncrementName**, to underline the fact that what we are delivering is business value.

For example, the **valueIncrementName** in product development could be "Product Increment". But **valueIncrementName** could be many other things in Enterprise Scrum like "Sales Increment", "Profit Increment", "Research Increment", "Compliance Increment", "Strategy Development Increment", "Service Operations Increment", "Product X Development Increment", etc.

In Enterprise Scrum we must be more general than Scrum, and we can't assume that the VLIs map directly to deliverables with **businessValue**; although this is ideal. Therefore we map how the VLIs map to deliverables through the parameter **VLIsToDeliverableMap.**

Metrics

In Scrum, the accepted **metric** is to measure the amount of "effort left to completion" measured in time, or relative units. But in Enterprise Scrum we can manage very many different things i.e. companies, sales, product development, operations, audit, etc.; so there needs to be a compatible set of measurements on well-defined metrics that allow us to monitor progress in achieving **businessValue**, and with the balance that we want to achieve in our **optimizationGoals** with the **optimizationVariables**, which are a good starting point to define an appropriate **metric**.

By **metric**, we mean a quantifiable operational measurement of progress that reports how much **businessValue** we are delivering. For example, our metric may have a **name** like “Real State Gross Sales”, with a **description** like “amount of gross sales left to achieve our sales goal”. It is important to specify the **unit** used to measure our **metric**.

Metrics in Enterprise Scrum can be given both in terms of “goals achieved” or “goals remaining”. Both are useful, but we prefer the “goals remaining” metric to drive things to completion or achieve a higher goal. Using these metrics, a burn-down, a burn-up, or a chart of a different type can be produced.

We may have one or more metrics, so it is convenient to keep a list of **metrics.**

Velocity

The canonical velocity is the aggregate size (effort) of ALL VLIs finished according to a DOD, and approved by the **businessOwner** in an Improvement Cycle. For example in development we would aggregate all the story points in an Improvement Cycle (Sprint). But in Enterprise Scrum we can use a **metric** for the **principalVLIType** and come up with a “generalized velocity” – that is, not the traditional “effort velocity”, but one that more accurately measures business value for our process.

For example, in internal control management, we would aggregate the number of controls tested (burnup) – not necessarily the effort that it took to test the controls. Or even better, the number of controls still be tested to achieve a certain compliance goal (burndown).

Another example. In sales, our velocity could be the added profits of all sales in that iteration (burnup), or the profits that are still remaining to achieve a profit goal (burndown).

Calculations

Because in Enterprise Scrum we can have a **listOfImprovementCycles** we can make predictions if the velocities of the Improvement Cycles have been measured.

This allows us to make **calculations** about the future like a **calculation** for budgets, schedules, or calculations about the past like global customer satisfaction levels and manage risk as we measure with the caveat that we need to re-estimate our plans

This doesn’t mean that we make “static future plans” in Enterprise Scrum. It only means that we can make the “best prediction” given what we know at some point in time. For example, we may calculate schedules, budgets, scope to be achieved by a certain date, customer satisfaction, total net sales, etc.

Schedules and Releases

The value list size is:
VL-Size = Σ **size (**from the **valueListItem** in the appropriate **sizingUnit)**

Number of improvement cycles:
ICs (improvement cycles) = VL-Size (Value List size)/ V (velocity in the appropriate **metric**)

Then we simply calculate time as:

Schedule = ICs * **length**

Where **length** is the **length** parameter in the Improvement Cycle.

Burn rates

The total cost is:
TC (Total Cost) = resources * average salaries * HR factor + proposal cost + other fixed costs

And therefore the burn rate is:
BR (Burn Rate) = TC/Number of Yearly Improvement Cycles

Budgets

Since we know the burn rate and the number of improvement cycles, we can then calculate the budget:

$ (Budget with no profit) = ICs * BR

Even with a profit percentage X:
$ (Budget with X% profit) = ((1 + X)/100) * $

We can do similar calculations for sales projects, net profit, cumulative customer satisfaction, etc.

Reports

In Enterprise Scrum, we use generic versions of the standard Scrum reports, but in addition we may use other **charts**.

We may use a customied **scrumBoard**, or **chart**, like a burndown to track *any*

Improvement Cycle e.g. Bi-weekly Business Management Improvement Cycle, or a Quarterly Business Management Improvement Cycle.

ScrumBoard

Because in Enterprise Scrum we have generic Improvement Cycles, the **scrumBoard** need to be customized to reflect this. They may also be recursive.

For example, a company may have 6 bi-weekly “Biweekly Improvement Cycles” into a larger “Quarterly Improvement Cycle”.

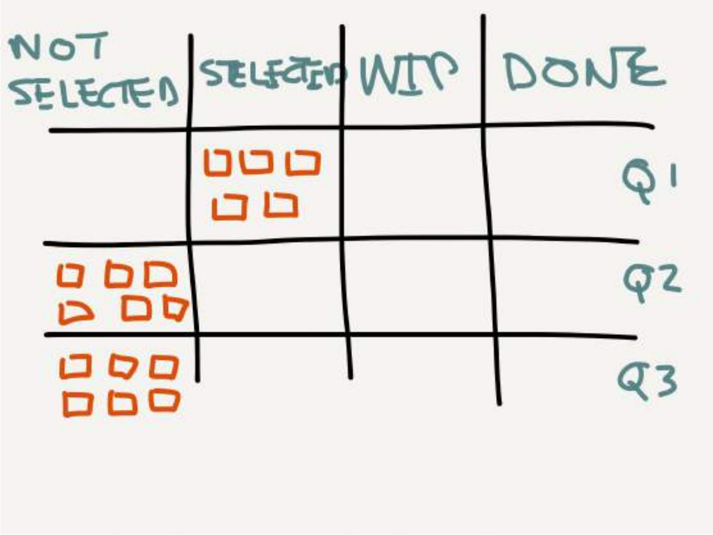


Figure 13. Scrum Board can be used for any Improvement Cycle length

within each quarterly row, there will be 6 bi-weekly columns like this one:

NOT SELECTED	SELECTED	WIP	DONE
1 Bi-weekly VLIs			
2 Bi-weekly VLIs			
3 Bi-weekly VLIs			
4 Bi-weekly VLIs 5 Bi-weekly VLIs			
6 Bi-weekly VLIs			

Figure 14. Nested Scrum Boards

Burndowns and other Charts

Burndowns are reports that tell us how much more we need to achieve our goals. The only burndown defined in the Scrum framework is the Sprint Burndown, which typically reports “hours left to completion” vs. Sprint length (1-4 weeks).

As in Scrum, the preferred progress report in Enterprise Scrum is a burndown that keeps track of size left to achieve our goal per unit of time, measured in **sizingUnit** through the appropriate **metric** – regardless of the nature or length of the improvement cycle.

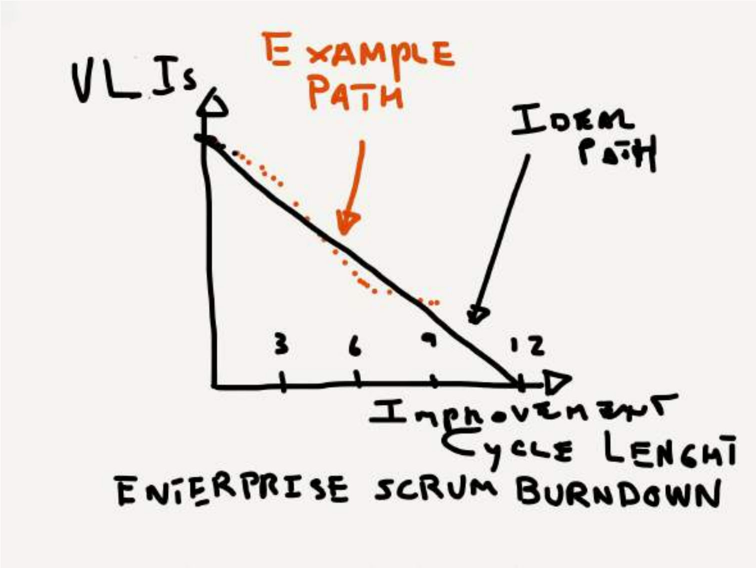


Figure 15. Burndown and other charts can be used for any Improvement Cycle

For example, we can build a **metricChart** with a **name** like “daily sales”, “cumulative sales”, “cumulative profits” or “customer satisfaction level”, through a time series to measure a relevant quantity related to **businessValue**. We can also provide the **description** of each one of these charts and specify what **type** of chart it is: burnup, burndown, or other. Some **charts** don’t need to be burnups or burndowns – they could just measure something important like “daily sales” or “customer satisfaction” to check we are operating at the desired levels of performance.

We may need several **charts** to keep track of all the **optimizationVariables** and to achieve our **optimizationGoals** – the desired balanced among the **optimizationVariables** represented typically by a **metric.**

Scaling

A **parent** Scrum Team is a Scrum Team composed of the Business Owner of each collaborator of the **parent’s** Scrum Team, with a Business Owner and Scrum Master of its own. We sometimes call this the Chief Product Owner, but this is not strictly necessary – identifying the Business Owner of a parent team it’s enough.

Pattern: aggregate collaborators identical metrics
A **parent’s** mission is to deliver the Total Value Increment, which is the total value aggregated by all the Value Increments produced by different Improvement Cycles and by different Scrum Teams. Therefore, we must aggregate all the value of all Value Increments for all the **contributors** of a **parent**.

A **parent** typically has calculations to compute the Total Value Increment that is accumulated by the **collaborators** metrics. If this is the main metric of success, we call the **parent’**s velocity *global velocity,* and the **contributors** velocity, the *local velocities*.

For example, if we know the individual *local velocities* of a collection of sales teams managed through Enterprise Scrum from a common lower-level metric for

improvementCycle like “Net Profit”; then we can calculate the parent’s *global velocity* – the net profit of all the sales teams as the sum of all net profits for the **contributors**.

This allows the **parent** organization to make global projections for the sales of all stores for a higher level Improvement Cycle, like a Quarterly Improvement Cycle from a Weekly Sales Improvement Cycle. Each of the sales teams, the **parent’s contributors**, can also make longer-term projections based on their own local velocities.

Pattern: collaborators always approve parent’s plans
However, if the parent makes a projection, the **contributors** within a **parent** hierarchy need to approve a global projection plan by validating it through their local velocities. This is a general pattern in Enterprise Scrum at any level of scale.

Pattern: Big Room
Ideally all of the team members in all the **contributors** would be present in the PE3R meetings of the higher-level **scrumInstance** except the Daily Scrums**.** We call this the “Big Room pattern” parent because the parent meets with the contributors through the “Big Room”. For example, in sales, it is usual to see a “Quarterly Sales Planning Meeting”, a “Quarterly Sales Review Meeting”, or even a “Quarterly Sales Retrospective”.

Pattern: Representative System
However, because of space and communications limitations, we sometimes need to send representatives to higher level PE3R meetings. We call this the Representative System pattern”. For the representative system to work at least two key things are needed: 1) the representative needs to be knowledgeable about what the team is doing, and is capable of doing, 2) the representative needs to be trusted and communicate well with the team. For example, in difficult software projects, the higher-level PE3R planning meeting (Release or Sprint) is broken down in three parts: 1) first the representative scans the value list to “eye ball” what the team can do, then 2) the representative brings it to the team for the team to fully examine the proposed value list for the team, and 3) the team finally approves the value list and this is communicated to the higher level improvement cycle planning.

Pattern: Look Ahead Big Picture Dependencies
The representatives are typically the **businessOwner** of all **contributors** and sometimes a technical person, when the business owner is not knowledgeable in an area. When the system has dependencies, like in software development, the representatives can provide a first-cut of the dependency management, and then let the individual Scrum Teams resolve disagreements of these dependencies through their individual PE3R meetings.

Pattern: Self-organized Coordination
We prefer that all teams are coordinated as needed through self-organization – each Scrum Team is self-organizing, autonomous and cross-functional; so let every team resolve its dependencies. Larman and Vodde have observed this as well [LeSS1], [LeSS2]. In this environment integration and testing drives dependency management and therefore who we need to work with on a day to day basis.

Pattern: Scrum of Scrums

When the different collaborators' teams don't have good working relationships to "just start working with each other", it is necessary to do coordination by doing so-called "Scrum of Scrums" meetings. The purpose of these meetings is like the Daily Scrums: getting Scrum Teams to work with each other by getting to know what each of the teams are doing and identifying dependencies among them. Our goal should be to eventually stop these meetings when enough collaboration and knowledge-share is taking place among the teams.

Pattern: Coincident Improvement Cycles in Hierarchy

It is a very good idea that the **contributors'** Improvement Cycles are coincident with each other for a variety of reasons: aggregation with parent, true completion of dependent VLIs, etc. And it is also ideal that the Improvement Cycles of the **parent** are coincident with the **contributors'** Improvement Cycles because that way the **parent** will not report on partially completed work by the **contributors**.

Pattern: combine same level Improvement Cycle meetings across contributors

We can also combine same level **improvementCycle** meetings, like review, retrospective and refinement, as long as all of the **businessOwners**, **stakeholders** and the collective team members of all teams are invited and fit comfortably in the space for the meetings, and the size of the meeting is manageable. However, if they don't we need to have separate meetings with the set of business owners that make up the **parent scrumInstance**.

Pattern: Communicate Up

When trying to accomplish some of the goals that we agreed upon at the Planning meeting, the team may find that either:

1) the team can't do VLIs with its current team members
2) that the VLIs are impossible to do (due to technology or domain constraints)
3) that some or all of the VLIs have a high uncertainty and they will not be complete by the end of the improvement cycle
4) that some or all of the VLIs are no longer needed
5) that some VLIs are needed but they were not planned for (or accepted in that improvement cycle)

All of these events need to be communicated right away to the Business Owner, so that the Business Owner can make the appropriate decisions.

Pattern: News Channel

Sometimes a team finds something interesting, like a team developing new medicines may find that they just discover a new vaccine or cure for a disease, or a new product. These news need to be communicated not only to the Business Owner, but possibly to other teams or organizations that want to listen to their "news channel". *This is one way by which leadership can come from any level in the organization.* This is specially a good pattern if we are trying to achieve a Multi-Agent organizational architecture.

Pattern: combine different level Improvement Cycle PE3R meetings
It is a good idea sometimes to combine different Improvement Cycle PE3R meetings when these meetings are coincident. For example, in the last week of a quarter, instead of having individual Review meetings for the Weekly Sales Improvement Cycle, just have one Review meeting for the entire Quarter. You must be able to demonstrate each of the sales teams performance as well as the overall performance to combine the meetings.

Improvement Cycle

As we said before, each **ImprovementCycle** has the same PE3R cycle, where the **parent's** PE3R **improvementCycle** is coincident with the **contributors** PE3R **improvementCycles**, and possibly having multiple nested improvement cycles. As we work and complete improvement cycles, we can save the ongoing and final results in the **improvementCycleInstances** of each of the **improvementCycles.**

Planning

Sprint Planning – part one, Agree with Business Owner what to do

The **businessOwner** and the team members get together so that the team can accept to attempt a certain amount **businessValue** carried by **valueListItems** from the **valueList**. Before any **businessValue** is accepted by the team, care should be taken to see that each **valueIncrementItem** has a **DOR** and a **DOD**.

If there is already a measured velocity – a **metric** that measures the **businessValue** of the **principalVLI** progress, then the **team** can use that velocity to accept work for an **improvementCycle** as long as the accepted VLIs don't pose any concerns. If they do, they the team may reject one of more VLIs, despite appearing to fit within the velocity.

If there is not a measure velocity, the team needs to estimate how much work the team can take based on their experience and then use one or more **improvementCycles** to calibrate their estimates. It is very likely that the estimate is be wrong. If the estimate is too small, the team can attempt more **businessValue** from the **valueList** as long as the Business Owner approves they can pull more work. If it is too much, the team will not complete the work estimated, and some of the attempted **businessValue** will not be achieved.

Either way, our team will know at the end of the **improvementCycle** how much businessValue is appropriate to attempt in the next Improvement Cycle.

In some cases, the **valueList** has **selectable valueListItems**. For example, a software development team can select what features to implement in a Sprint. However, in some cases, the **valueListItems** are non **selectable**, for example, in the case of retail sales items.

Our goal in Enterprise Scrum is to deliver true **businessValue**. However, sometimes **businessValue** cannot be achieved and the best option then is to deliver

potentiallyUsableValue – **businessValue** that would be releasable if the contents of the **valueIncrementName** would be sufficient to be delivered to a customer. In Scrum, we call this PSP, or potentially shippable product.

The result of this activity should be the **initialValueListForImprovementCycle**.

Sprint Planning – part two, Volunteer for VLIs, make plans for the VLIs, provide estimates for plans

Once the **team** agrees to attempt a certain amount of **businessValue**, the **team** members can then volunteer for **valueListItems**. The volunteers then provide a **planForVLI** for each **valueListItem** they agreed to attempt, and they provide estimates for the **planForVLI**.

Sometimes, it is a good idea to have a **VLIsToDeliverablesMap**, to specify the exact format of delivery of the **valueIncrementItems**.

For example, a **valueIncrementName** (Product Increment) with a given **potentiallyUsableValue** (Potentially Shippable Product) may be achieved by a software development **team** implementing **valueListItems** where the **planForVLI** is just a set of tasks (evaluated in hours), by a set of **volunteers**.

It is mandatory that the Business Owner stays with the team through the completion of this last planning stage, so that any doubts can be resolved by the Business Owner.

Execution

We assume that all the team members will use all the BA elements: collaboration, cooperation, helping each other and sharing knowledge; as well as the Scrum Values in their day to day activities.

Daily Scrums

The default Daily Scrum in Enterprise Scrum has the same format as in Scrum:

1) What did they do since the last time we met?
2) What will they do until the next time we will meet?
3) What issues or impediments the currently have
4) (Implicitly), what are the new estimates to completion for each of the remainder work they have, if applicable, only for estimable tasks, see Value List.

But the questions could be different. For example:

1) who do you depend on or could use help from?
2) Who depends on you or who can you use your help?
3) Issues or impediments

4) Estimates to completion

In some advanced teams, typically working on an Open Workspace environment, self-organization, collaboration and knowledge sharing is so high, that the Daily Scrums are no longer needed – everyone knows who they need to be working at an time.

The **team** members in Enterprise Scrum report on ***any type of work***. As in Scrum, the Daily Scrums are not only for status to see how everyone is doing independently, but so that dependencies one **dependsOnVLIs** that can be discovered on a daily basis and the team members can then better optimize their work for that day.

One or more **metricInstances**, **chartInstances**, **scrumBoardInstance**, **dependencyMatrix** or **calculationsInstances** could be updated on a daily basis.

The **chartInstances** can be burndowns, burnups, or other **charts**, of measured **metricInstances**, and can be updated on a daily basis to see how progress is made throghtout the **improvementCycle**.

The **scrumBoardInstance** is updated throughout the day as needed – not necessarily at the Daily Scrum.

We can keep track of the **dependsOnVLIs** through a **dependencyMatrix** for an **improvementCycle** either for the VLIs within a team, for VLIs in other teams, or for individuals or teams.

So for example, the Company Management **scrumInstance** can report on work for business-related issues; the Portfolio Management **scrumInstance** can report on portfolio or program level work; and the Development **scrumInstances** can report on their development work.

In the Scrum community, we call a higher-level Daily Scrum a Scrum of Scrums. In Enterprise Scrum we simply call it the Daily Scrum for that **scrumInstance**.

The **doneValueListForImprovementCycle** are completed according to their **DOD.** The **businessOwner** can approve of the completion of the different VLIs, so that the team can attend the review with confidence that the **businesOwner** is satisfied with the results.

Review

The review for an **improvementCycle** has the purpose of showing all the **businessValue** accomplished through the Value Increment by the team to the **businessOwner** and other **stakeHolders**.

It is ideal that the **businessOwner** has been looking and approving the done VLIs within the **improvementCycle**, so that the review has the form of a celebration and showcase the **businessValue** accomplished.

Retrospective

The retrospective is done to provide an opportunity for the **scrumTeam** that owns that **scrumInstance**: the **businessOwner**, the **stakeHolders**, the **scrumMaster** and the team members; to provide feedback for improvement about either the **businessValue** accomplished, the Scrum process itself, or the collaboration of the **scrumTeam**.

The retrospective typically has the format of first providing feedback on things the Scrum Team is doing well; and then providing feedback on how to improve.

The results of the retrospective are a list of what the Scrum Team is doing **well**; and a list of **improvements** that if possible need to be implemented right away: our first goal is to improve. It's a good idea to keep the results of the retrospective for future **improvementCycles**; that way the conversation is continued, not started again i.e. the wheel doesn't have to be reinvented again.

Value List Refinement

The **businessOwner** which is always interacting with the **stakeHolders** can always refine – re-prioritize, add, change, the **valueList**. But the refinement is the official meeting that is the last opportunity before a new **improvementCycle** is started to change the **valueList**. The same activities that are needed for the initial **valueList** are needed for refinement: definition, priority, colorizing, and sizing.

more Improvement Cycles

Once an **improvementCycle** is finished, a new **improvementCycle** is started to accomplish more **businessValue**. This can happen at multiple levels or just at one level.

For example, a business management team, can monitor the accomplishment of business goals through a 2-week Improvement Cycle. When one improvement cycle finishes they can start another 2-week Improvement Cycle. However, the business team can also be doing a quarterly 6-week Improvement Cycles, and using the 2-week Improvement Cycle to closely monitor goals and make adjustments. The end of the 6-week improvement cycle is coincident or synchronized with the 2-week improvement cycles. These improvement cycles could include one or more teams, or course.

3. Enterprise Scrum Example -- Real State sales

Scrum Instance description

The Enterprise Scrum definition tells us about all the options for customizing Enterprise Scrum as a general management system.

But how does a customization look like? **This is just a short example to get the reader a feel for Enterprise Scrum**.

If we try to manage a real-state sales operation through Scrum we may define a Product Backlog item as “a property to be sold”, *but we may NOT know exactly which ones will sell*. There are other problems. The PBIs (product backlog items) of the Product Backlog – wait, we are not building a product, are sized in effort, but in sales we are really not interested in estimating the effort of a sale – we are interested in profit, revenue, and other business-like things. When we make longer-term projections, we are not interested in making projections about cost of sales– we would like to make “sales projections” in revenue or profit. Finally, we might be interested in tracking more than one quantity, not just profit or revenue, but maybe customer satisfaction, o product quality.

Since we don’t know in advance which properties exactly will be sold, and we are not interested in “sales cost or effort”, and we want to make projections of sales – not cost; we may be tempted to say that Scrum can’t be used to manage our sales process because it is incompatible with regular Scrum.

However, people around the world have found and find almost daily, a way to use Scrum to manage a myriad of processes by making ad hoc adaptations. Enterprise Scrum is a collection of those adaptations so that we can use Scrum to manage almost anything in an agile way.

Scrum Team Parameters

So let’s customize the real-state sales for a fictitious company name “Agile Realtors Inc.”, that has 3 different sales teams sometimes interacting with each other:

the Residential Real-State Sales team,
the Vacation Property Sales team, and
the Commercial Real-State Sales team

These names are the **instanceNames** of our Scrum Teams. We would then have to specify who are the **businessOwners**, **stakeHolders**, **scrumMaster** and **team** for those Scrum Team.

Architecture Parameters

We are doing sales – so we might be tempted to say that we don’t need architecture parameters, right? Well, actually we do.

Architecture studies the relationship of concepts, so in real-state sales, not every sales person knows the relationships of all concepts and to make a sale. In fact, it is typically just the senior sales people that know these relationships well enough so that they can mentor the junior sales people into them.

What might these relationship be?   Appraisals, inspections, city codes, additions, blueprints, sales commissions structures, taxes, fees of all kinds, real state agreement client contractual clauses, buyer's or seller's contractual clauses, banking rules and procedures, applications, offers, credit reports, etc. – the relationship of all of these concepts is what we could call the "architecture of real-state sales". So **architectureType** is "real state sales".

Depending on our team, we might find or choose we could use a self-organizing **architecturalManagementType,** this is very desirable in Scrum teams of any kind, because that way there won't be any strong knowledge dependencies among the team members. However, in reality this may be difficult to find or to achieve on day one. It may take some coaching and a lot of learning before we get there. So let's assume that the **architecturalManagementType** is by "Agile Architect", in other words, someone with knowledge within the team will be mentoring and *checking the relationship in realtime.* If the relationships were just checked at the beginning or end of an Improvement Cycle, then this would look more like Traditional Architect style of management.

Business Parameters

Our **domain** of course is Sales for all 3 teams.   For all of them the **businessCycleType** is operations – because we are not developing a product or service, we are managing operations.

But what exactly is **businessValue** for a sales team? We will define "net profit" to be our number one goal for business value. However, we want to maximize net profit provided we don't compromise customer satisfaction too much, and that we don't have sales of substandard properties. I just provided a **description** of what **businessValue** in terms of the **optimizationVariables**: net profit, customer satisfaction and product quality. I can further specify optimizationGoals such as: we want at least 99% customer satisfaction, and 99.55% product quality (no problems or complaints with the properties sold after the sale.)

Template Parameters

We could realize that there is already a **template** for Real State Sales, and choose to use one. But for the example here, let's pretend one is not yet available.

Structure Parameters

The Agile Realtors Inc. company is also Scrum managed – meaning there is an upper management Scrum Team for the company “Agile Realtors Inc.” that serves as the **parent** for all the sales teams defined above and some additional support teams like HR and accounting.

The **parent** may have as its **team** members all of the Business Owners of the **contributors –** this is customary, and some additional individual members. Let’s assume that HR and Accounting are also Scrum teams but that they are not scaled at all: the HR team has only one person -- the business owner, and that accounting has only 3 people plus the business owner.

Therefore, we would have 5 **team** members but only 4 **contributors**. This helps us identify where the organization is scaled.

The company management team – and any Scrum Team at any layer for that matter, *can do Scrum regardless of the depth of scaling underneath.*

If there were complex relationships we could map who we **dependsOn** on who **dependsOnUs**. Let’s say that the Vacation Property Sales team is growing very fast, so we could specify a dependency with the HR team. This relationship may be temporary. In general we can track our dependencies with a dynamic **dependencyMatrix.**

Technique Parameters

Let’s say that we want to standardize on the sale method by Sharon Drew Morgen called Buying Facilitation. This is the **name** of the **technique** we want to use. We want to use it for the **purpose** of selling more with more customer satisfaction. We would need to identify what **processStep** is it used: in the execution of an **improvementCycle**, and described **howItIsUsed**, and how it is mapped to Scrum through the **mappingsToScrum** as well as some **references**.

> ***Enterprise Scrum is a formal process framework: we can insert any number of techniques to customize its behavior.***

Value List Parameters

We had defined in detailed what **businessValue** is for our sales operation, so it is really easy to see what our **principalVLIType** is “real state properties”.

There might be other VLI-types: legal, contracts, bank-related, credit reports, sales tools, customer contract signups, accounting-related, etc. We would list all of them as **VLItypes**.

We could have our company pushing some properties more than others and have our sales **prioritizable**, but we are going to choose not to do this. Therefore, our **orderingTechnique** would be blank at this point. This is ok. Not all parameters need to have values.

Our **sizingUnit** will be "revenue from sale" – this is customary in a sales process. We are very certain about the **sizeCertainty**, so we are going to say 90%. This number is just going to help us gauge how certain is to make projections and forecasts.

Real-State sales are for the most part independent, so we are going to say that the **dependentVLIs** are NOT dependent. However, the knowledge to sell is still important to share, so in our Daily Scrums we still want to find out who we need to collaborate with.

Our properties are listed as our **valueListItems** in our **valueList,** as well as other infrastructure stuff that sales are dependent on -- this is typical in Scrum: not everything we do is gravy, there is unavoidable infrastructure or support VLIs that we need to do and keep track of.

We would also need to have a common **DOR** (Definition of Ready) for all properties – not all properties are ready to be sold even if they are under contract: they would have to have a number of things to be "available for sale": appraisals, inspections, comparable analysis, etc. This would be another customization on the Scrum process – beyond and above that in regular Scrum.

Our **DOD** (definition of done) for a sold property, would be to: 1) have that item sold with a signed contract, 2) secure payment for the transaction, and 3) ensure the payment is valid and cleared in a business checking account. With this **DOD** we can make sales revenue **charts**, for example a burndown of "properties to be sold per our definition of done to match a target", and account for our "Sales Increment", which in this case is our **valueIncrementName**. We would track down and update our revenue burn down during the Sprint – maybe even allowing for extra properties to be sold if we found a way to improve our sales process.

Value List Instance Parameters

Each one of our VLIs (**valueListItems**) – each one of the properties, will have a **name**, a **description**, no **priority –** because we said our VLIs were not **prioritizable, size** in revenue dollars. Each property will inherit the **DOR** and **DOD** from the VLI-type. However, in Enterprise Scrum, as in regular Scrum, each VLI (PBI), needs to have a definition of done.

Because we have a sales methodology in place, our **planForVLI** could be standard.

In the particular case of real state sales, it is rare to have a **parentVLI** – maybe through selling an entire condo building, or a subdivision. And neither will it have a **depndsOnVLIs** – because real-state sales don't depend on each other.

We also have to specify if the VLI or VLIs are **selectable**, meaning, can we select a real-state property into an improvement cycle, and have certainty that it would be sold? Not, of course not. Therefore, our "sales velocity" – the velocity of our **principalVLIType**, is going to be based on the **metric** "sales revenue", but not the specific properties to be sold.

Additional attributes can be added for the VLI: who added the property, what sales budget does it have, etc.

Improvement Cycle Parameters

Our company management team, has decided we will have 3 overlapping nested **improvementCycles**:

1 week ”Weekly Sales Improvement Cycles”
3 month “Quarterly Sales Improvement Cycles”
1 year “Yearly Sales Improvement Cycles”

Each of these improvement cycles has a PE3R structure: planning, execution, review, retrospective and refinement. It is best that these cycles are coincident for each Scrum Team, and for all Scrum Teams as well.

When we are start or when we finishing a higher-level improvement cycle, we could skip the lower-level improvement cycle as long as we can invite everyone involved to the higher-level improvement cycle. We call this the “Big Room” pattern. For example, we could have a merged quarterly planning, review or retrospective across all teams, instead of having the individual weekly planning, review and retrospectives and then go to the quarterly reviews to review the same information. However, if we can’t fit everyone, or there are other logistical problems, we may want to use the “Representative System” pattern, and then have the weekly sales reviews and retrospectives, and then send a representative to the Quarterly Sales Review meeting.

Each improvement cycles is defined with a **name**, **level** (counting from the day), **length** (in time), **metrics (“**sales revenue” for all sales teams and all improvement cycles), **charts** (burndown of revenue to be sold), **scrumBoard** types (using the workflow of the Buying Facilitation **technique** at the weekly level, but just pipeline statistics for Quarterly sales), no **dependencyMatrix** since real-state sales are not dependent on each other. The name our **valueIncrementName** is “Sales Increment”, of course.

We could also have a number of **calculations** that are important for the company at the weekly level for each team. For example, total revue across all teams, net profit, customer satisfaction (from survey), property quality (from complaints and after sales interiews/surveys), etc.

We can also have the company management team *aggregate compatible metrics for the quarterly reports for all teams.*   This is a useful pattern to estimate *global velocity* from local velocities and establish global **metrics** and **charts from** local **metrics.** As such, since we have measured a Quarterly Sales Revenue before, we could build expectations for sales revenue in the future.

In other words, we can use the concept of global and local velocity for any improvementCycle, and scale with that our expectations in time and structure.

Improvement Cycle Instance Parameters

Once we define our improvementCycles, we can have improvementCycleInstances of them – this is where we track the day to day specifics of each improvementCycle, and then record what happened in them for future reference.

For example, in the planning meeting of our “second week of the first quarter” -- this is the **icID** of the **improvementCycleInstance**, the sales teams would accept a weekly sales target based on the “first week of the first quarter” velocity - i.e. on what the team sold last week. This is the **metricInstance** that resulted from the first week.

We also record what was the **initialValueListForImprovementCycle**, and what was actually done according to our DOD, **doneValueListForImprovmentCycle**.

During Spring Execution, the sales team would get together every morning and meet at the Daily Scrum to answer the 3 usual questions: 1) what did you work on? 2) what will you work on? 3) what issues or impediments do you have? Real-state Sales are typically independent of each other, so the purpose is not to identify dependencies. *Instead, the purpose of the Daily Scrum meetings, is to help each other with sales tactics, knowledge, understanding of situations, and occasionally pick up work from each other – someone could be going on vacation, or be sick, and other people can pick up their work if needed to be.*

Every day we could update our **metricInstances**, **chartInstances**, **scrumBoardInstance** and **dependencyMatrixInstance**.

In our burndown chart, we will show target revenue in the Y-axis and time in the X-axis, as usual. Profit could also work as a generalized velocity, but most people choose revenue to manage their sales process. The team members would update their sales to show in the “target revenue burndown” how their cumulative numbers are looking every day. This chart should be updated every day and displayed visibly. The Scrum Board will show all the properties in process sometimes with customized workflows, in this case from our Buying Facilitation **technique**, in the WIP and DONE columns. When the Sprint is done, all of the properties in progress will automatically move to the next Sprint because we are not prioritizing their sales into any one Sprint.

At our review meetings, when the Sprint time-box is done, the teams will show how much progress they made individually and review how they achieved their sales goal. The meeting can also be structured as a mini-celebration with food. The Business Owner, the sales owner, should be there to review the overall results as well as being involved as much as possible in the day to day operations.

At the Retrospective, the team should get together and ask the 2 familiar questions: 1) what are we doing well? and 2) how can we improve? The results of the **retrospective** are recorded with what we are doing **well** and what **improvements** we want to put in place.

First each of the team members – one by one, should write down and post on a board what they think they are doing **well**, to avoid duplicates. And then they should do the same with ideas for improvement. Maybe they have a better way to interact with the banks, or they have techniques for ensuring pre-qualifications for the buyers, etc. Finally, the team members and the Business Owner should choose which **improvements** to put in place, and keep the results of the retrospective, and bring them to the next retrospective in order to agilize the process.

Finally, at refinement, the Business Owner or each team should provide guidance as to what type of properties desires to sell or what type of transactions the firm would rather do, etc..

The weekly **improvementCycle** process starts again and again for each week, and builds expectations for the quarterly **improvementCycle**, where aggregates are calculated across the 3 sales teams and compared to expectations based on empirical evidence.

This is an example of how to configure and use Enterprise Scrum.

References

1. [AgileAtlas], http://agileatlas.org/atlas/scrum
2. [AgileCompetitors] Steven L. Goldman; Roger N. Nagel; Kenneth Preiss. Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer (Kindle Locations 205-206). Kindle Edition.
3. [AgileManifesto], www.agilemanifesto.org 2/11/2001.
4. [AgileSoftwareDevelopment] Beedle M., Schwaber K., Agile Software Development with Scrum, Prentice Hall, 2001.
5. [Alexander] Alexander, Christopher (1979). *The Timeless Way of Building*. Oxford University Press. ISBN 978-0-19-502402-9.
6. [ArthurDLittle] Arthur D. Little, Innovation Excellence Study, 2012.
7. [Atkins] Lyssa Atkins, Coaching Agile Teams: A Companion for ScrumMasters, Agile Coaches, and Project Managers in Transition, Addisson and Wesley, 2010.
8. [BalancedScoreCard], Kaplan, Robert S.; Norton, David P. (1996-08-02). The Balanced Scorecard: Translating Strategy into Action (Kindle Location 3). Perseus Books Group. Kindle Edition.
9. [Beedle-cOOherentBPR-1997] cOOherentBPR: A pattern language to build Agile organizations, Michael A. Beedle, PLoP '97 Proceedings, Tech. Report #wucs-97-34, Washington University (1997).
10. [Beedle-EnterpriseArchitecturePatterns-1998] Enterprise Architecture Patterns: Building Blocks of the Agile Company, Michael A. Beedle, SIGS, New York, (1998).
11. [BeyondBudgeting] Jeremy; Fraser, Robin (2003-02-25). Beyond Budgeting: How Managers Can Break Free from the Annual Performance Trap . Harvard Business Review Press. Kindle Edition.
12. [BlueOcean], K. Chan and Mauborgne R., Blue Ocean Strategy – how to create uncontested Market Space and Make a the Competition Irrelevant
13. [Brooks1], R. A. Brooks (1987). "Planning is just a way of avoiding figuring out what to do next", Technical report, MIT Artificial Intelligence Laboratory.
14. [Brooks2] R. A Brooks (1991). "Intelligence Without Representation", Artificial Intelligence 47 (1991) 139-159.
15. [BusinessModelGeneration] Osterwalder, Alexander; Pigneur, Yves (2013-02-01). Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers (Kindle Location 361). Wiley. Kindle Edition.
16. [BusinessModelYou] Business Model You: A One-Page Method For Reinventing Your Career (Kindle Locations 411-412). John Wiley and Sons. Kindle Edition.
17. [CEK] Cooper, Edgett and Kleinschmidt*, Best Practices in Product Innovation: What Distinguishes Top Performers* by Cooper, Edgett and Kleinschmidt, 2011.
18. [Christensen], Christensen C., The Innovator’s Dilemma: when New Technologies Cause Great Firms to Fail, Harvard Business Press, 1997.

19. [Cohn1] Cohn, Mike (2004-03-01). User Stories Applied: For Agile Software Development (Kindle Locations 269-270). Pearson Education (USA). Kindle Edition.
20. [Cohn2], Cohn M., Agile Estimating and Planning, Prentice Hall, 2006.
21. [Cohn3], Cohn M., Succeeding with Agile: software development using Scrum, Addison-Wesley, Upper Saddle River NJ, 2010.
22. [Collins1], Collins J. Porras, J., Built to Last, Harper Collins, Collins Business Essentials, 1994
23. [Collins2], Collins J., Good to Great: why some companies make the Leap… and other don’t?, Harper Collins, 2001
24. [Coplien] James O. Coplien, Borland Software Craftsmanship: A New Look at Process, Quality and Productivity, Software Production Research Department, AT&T Bell Laboratories, Proceedings of the 5th Annual Borland International Conference, Orlando, Florida, 5 June 1994
25. [DAD] Scott Ambler, Scott Lines, Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise, IBM Press, 2012.
26. [DesignThinking], Kelley T. Littman Jonathan, The Art of Innovation, DoubleDay, 2000
27. [Drucker1], Drucker, Peter (1957). Landmarks of Tomorrow, New York: Harper & Row. pp. 122. ISBN 978-1-56000-622-0.
28. [DesignThinkng], Brown T, Change by Design: how design thinking transforms organizations and inspires innovation, Harper Collins, 2009.
29. [DistributedScrum] Woodward, Elizabeth; Surdek, Steffan; Ganis, Matthew (2010-06-21). A Practical Guide to Distributed Scrum (Kindle Location 286). Pearson Education (USA). Kindle Edition.
30. [Eckstein] *Eckstein, Jutta; Agile Software Development in the Large: Diving Into the Deep (Dorset House eBooks) (Kindle Locations 11-12). Pearson Education. Kindle Edition.*
31. [Ford1], Ford, Henry; Crowther, Samuel (1930). Edison as I Know Him. Cosmopolitan Book Company. p. 15 (on line edition).
32. [GameStorming] Gray, Dave; Sunni Brown; James Macanufo (2010-07-21). Gamestorming: A Playbook for Innovators, Rulebreakers, and Changemakers (p. 1). OReilly Media - A. Kindle Edition.
33. [Hamel1] Hamel, Gary; Prahalad, C. K. (1996-03-21). Competing for the Future, Perseus Books Group.
34. [Hamel2], Hamel G., What matters now: how to win in a world of relentless change, ferocious competition, and unstoppable innovation, Jossey-Bass (Wiley Imprint), San Francisco CA, 2012
35. [Hammer], Michael Hammer and James Champy, Reengineering the Corporation, Harper-Collins, New York, 1993.
36. [HBR-ChangeManagement] Harvard Business Review (2011-02-24). HBR's 10 Must Reads on Change Management (including featured article 'Leading Change,' by John P. Kotter) (Kindle Location 977). Perseus Books Group. Kindle Edition.

37. [Hoshin1], Hoshin kanri for the lean enterprise : developing competitive capabilities and managing profit / Thomas L. Jackson, New York : Productivity Press, c2006.
38. [Hoshin2],, Hutchins, David (2012-09-01). Hoshin Kanri (Kindle Locations 4-6). Ashgate Publishing. Kindle Edition.
39. [LeSS1], Larman C. Vodde B., Scaling Lean and Agile Development: thinking and organizational tools for large-scale Scrum, Addison and Wesley, Upper Saddle River NJ, 2009.
40. [LeSS2] Larman, Craig; Vodde, Bas (2010-01-26). Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum (Kindle Location 10). Pearson Education (USA). Kindle Edition.
41. [LeanStartup], Reis E., The Lean Startup: how to make Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, New York, 2011.
42. [Lefingwell] Leffingwell, Dean (2007-02-26). Scaling Software Agility: Best Practices for Large Enterprises (Kindle Location 10). Pearson Education (USA). Kindle Edition.
43. [Liker1], Liker J., The Toyota Way – 14 management principles from the world's Greatest Manufacturer, McGraw-Hill, New York, 2004
44. [Liker2], Liker J. Morgan J., The Toyota Product Development System: integrating people, process and technology, Productivity Press, New York, 2006.
45. [LittleBets] Sims, Peter, Little Bets: How Breakthrough Ideas Emerge from Small Discoveries. Free Press. Kindle Edition. (2011-04-19).
46. [Nagel1], Nagel R., –21st Century Manufacturing Enterprise Strategy, Roger Nagel, Iacocca Institute, Lehigh University, 1991
47. [Kanter], Kanter, Rosabeth Moss (14 June 2011). "Innovation: the classic traps". *Harvard Business Review on Inspiring and Executing Innovation*. Harvard Business Press. pp. 149–181. ISBN 978-1-4221-6261-3.
48. [Kaplan] Kaplan, Saul, The Business Model Innovation Factory: How to Stay Relevant When The World is Changing. John Wiley and Sons. Kindle Edition.
49. [Katzenbach] Jon Katzenback, Dougas K. Smith, The Wisdom of Teams: Creating the High-Performance Organization, Harper Collins, 2006.
50. [Kniberg] Kniberg, Henrik, Lean from the Trenches: Managing Large-Scale Projects with Kanban (Kindle Location 185). Pragmatic Bookshelf. Kindle Edition.
51. [LeadingChange] Kotter, John P. (1996-08-07). Leading Change (Kindle Location 83). Perseus Books Group. Kindle Edition.
52. [KotterForbes] Can you handle an exponential rate of change? http://www.forbes.com/sites/johnkotter/2011/07/19/can-you-handle-an-exponential-rate-of-change/
53. [Management3.0] Jurgen Apello, Management 3.0: Leading Agile Developers, Developing Agile Leaders, Addison and Wesley Professional, 2011.

54. [MMM], Brooks F., The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley, Boston, 1974, 1995.
55. [MultiAgent] Wooldridge, Michael (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons.
56. [Nonaka-PragmaticStrategy] Nonaka, Ikujiro; Zhu, Zhichang. Pragmatic Strategy (Kindle Location 188). Cambridge University Press. Kindle Edition.
57. [Nonaka-ManagingFlow], Nonaka, I., Toyama, R. and Hirata, T. 2008. Managing Flow: A Process Theory of the Knowledge-Based Firm. New York: Palgrave Macmillan.
58. [Nonaka-KnowledgeCreating] Ikujiro Nonaka; Hirotaka Takeuchi. The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation (Kindle Location 91). Kindle Edition.
59. [NonakaTakeuchi], Takeuchi, Hirotaka; Nonaka, Ikujiro (January–February 1986). "The New Product Development Game" (PDF). Harvard Business Review.
60. [Nonaka-KnowledgeCreationAndManagement] Ichijo, Kazuo; Nonaka, Ikujiro (2006-11-08). Knowledge Creation and Management:New Challenges for Managers (Kindle Locations 20-21). Oxford University Press. Kindle Edition.
61. [Ohno], Ohno, Takechi, Toyota Production System: Beyond Large-Scale Production, Productivity Press, Portland OR, 1978.
62. [OrgPatterns] James Coplien, Neil Harrison, Organizational patterns,
63. [Peters] Peters, Tom (2010-09-08). The Circle of Innovation: You Can't Shrink Your Way to Greatness (Vintage) (Kindle Location 80). Random House, Inc.. Kindle Edition.
64. [Pink1], Pink, Daniel H. (2011-04-05). Drive: The Surprising Truth About What Motivates Us (Kindle Locations 77-78). Riverhead Books. Kindle Edition.
65. [Pink2], Pink, Daniel H. (2006-03-07). A Whole New Mind: Why Right-Brainers Will Rule the Future . Penguin Group. Kindle Edition.
66. [Poppendieck], Poppendieck M. Poppendieck T., Lean Software Development: an Agile Toolkit, Addison and Wesley, 2003
67. [Porter1], Porter M., Competitive Advantage: creating and sustaining superior performance, The Free Press, 1985 1998.
68. [Porter2], Porter M., Competitive Strategy: Techniques for Analyzing Industries ad Competitors, The Free Press, 1980, 1998.
69. [Peopleware], Tome DeMarco and Tim Lister, Peopleware: Productive Projects and Teams, Dorset House, 1987.
70. [PWC], Price Waterhouse Coopers, “15th Annual Global CEO Survey 2012,” available at http:// www.pwc.com/ gx/ en/ ceo-survey/ pdf/ 15th-global-pwc-ceo-survey.pdf.
71. [PowerofScrum] Sutherland, Jeff; van Solingen, Rini; Rustenberg, Eelco (2012-01-31). The Power of Scrum. . Kindle Edition.
72. [ProcessControlTheory], Ogunnaike Babatunde A. and Harmon Ray W., Process Dynamics, Modeling and Control, Oxford University Press, 1994.

73. [ProfitZone], Slywotzky, Adrian J.; Morrison, David J.; Andelman, Bob (2007-12-18). The Profit Zone: How Strategic Business Design Will Lead You to Tomorrow's Profits (Kindle Location 127). Random House, Inc.. Kindle Edition.
74. [RadicalManagement], Stephen Denning, The leader's guide to radical management : reinventing the workplace for the 21st century, John Wiley & Sons, Inc. All rights reserved. Published by Jossey-Bass A Wiley Imprint 989 Market Street, San Francisco, CA 94103-1741.
75. [Rawsthorne] Dan Rawsthorne, Scaling Scrum with Scrum, https://leanpub.com/PPSAD
76. [RisingManns] Rising, Linda Ph.D.; Manns, Mary Lynn Ph.D. (2004-10-04). Fearless Change: Patterns for Introducing New Ideas (Kindle Location 165). Pearson Education (US). Kindle Edition.
77. [RunningLean] Maurya, Ash (2012-02-24). Running Lean: Iterate from Plan A to a Plan That Works (Lean (O'Reilly)) (Kindle Location 2). O'Reilly Media. Kindle Edition.
78. [SAFe] Scaled Agile Framework e, http://scaledagileframework.com/
79. [ScenarioPlanning], Wade, Woody (2012-03-14). Scenario Planning: A Field Guide to the Future (Kindle Location 59). John Wiley and Sons. Kindle Edition.
80. [Schiel] James; Schiel (2012-05-14). Enterprise-Scale Agile Software Development (Applied Software Engineering Series) (Page 19). CRC Press. Kindle Edition.
81. [Schliep] Andreas Schliep, Scaled Principles, http://www.scaledprinciples.org
82. [Schwaber] Ken Schwaber, The Enterprise and Scrum, Microsoft Press, 2007.
83. [SchwaberSutherland] Schwaber, Ken; Sutherland, Jeff (2012-03-23). Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust (p. 4). John Wiley and Sons. Kindle Edition.
84. [ScrumGuide], Jeff Sutherland and Ken Schwaber, Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game, Oct 2011.
85. [ScrumPLOP] http://www.scrumplop.org
86. [Stacey], Ralph D. Stacey, Complexity and Creativity in Organizations, Berrett-Koehler Publishers; 1 edition (January 15, 1996).
87. [SmartTribes] Comaford, Christine (2013-05-30). SmartTribes: How Teams Become Brilliant Together (p. 233). Penguin Group US. Kindle Edition.
88. [StandishGroup] Standish Group, The Chaos Manifesto, 2012.
89. [StoberHansmann] Stober, Thomas; Hansmann, Uwe (2009-11-19). Agile Software Development: Best Practices for Large Software Development Projects (Kindle Location 5). Springer. Kindle Edition.
90. [Sutherland] Jeff Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time, Random House, 2014.
91. [Swarm] Swarm Behavior, http://en.wikipedia.org/wiki/Swarm_behaviour

92. [TWI] Training Within Industry, http://en.wikipedia.org/wiki/Training_Within_Industry.
93. [TribalLeadership] Logan, Dave; King, John; Fischer-Wright, Halee (2012-01-03). Tribal Leadership: Leveraging Natural Groups to Build a Thriving Organization (Kindle Locations 395-398). HarperCollins. Kindle Edition. [Wicked], Peter DeGrace, Leslie Hulet Stahl, “Wicked Problems, Righteous Solutions: A Catolog of Modern Engineering”, Prentice Hall, 1990.
94. [Wommack1], Womack J. Jones D. Roos D., The Machine that Changed the World, Free Press, New York, 1990, 2007.
95. [Wommack2], Womack J. Jones D., Lean Thinking: Banish Waste and Create Wealth in your Corporation, 2nd Edition, Free Press, New York, 1996, 2003.
96. [Wommack3], Womack J. Jones D., Lean Solutions, , The Free Press, 2005